TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

METAMODELOVANIE, MODELOVANIE, VERIFIKÁCIA A VALIDÁCIA POŽIADAVIEK

Diplomová práca

2020 **František Ferenčík, Bc.**

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

METAMODELOVANIE, MODELOVANIE, VERIFIKÁCIA A VALIDÁCIA POŽIADAVIEK

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra počítačov a informatiky
Školiteľ:	doc. Ing. Zdeněk Havlice, CSc.
Konzultant:	doc. Ing. Zdeněk Havlice, CSc.

2020 Košice **František Ferenčík, Bc.**

Abstrakt v SJ

Táto diplomová práca poukazuje na možnosť uľahčenia a zefektívnenia inžinierskych prác naprieč celým životným cyklom softvérového systému od jeho počiatočného návrhu cez implementáciu až po údržbu vrátane verifikácie a validácie požiadaviek na tento systém. Zameriava sa na špecifickú oblasť softvérových systémov, ktorou je oblasť webových aplikácií založených na architektúre MVC (návrhový vzor interaktívnych aplikácií oddeľujúci biznis logiku od výstupu). Cieľom je navrhnúť metodológiu modelovania kritických častí tejto kategórie systémov. Táto metodológia je implementovaná ako CASE nástroj (počítačom podporované softvérové inžinierstvo). Z navrhnutej metodológie vznikne tento CASE nástroj vytvorením DSML (jazyk doménovo špecifického modelovania) s využitím metamodelovania a rozšírenia platformy generického modelovania WebGME. Na záver je metodológia a CASE nástroj demonštrovaná na príklade reálnej aplikácie igc2pdf určenej na generovanie letovej dokumentácie.

Kľúčové slova v SJ

modelovanie, metamodelovanie, validácia požiadaviek, verifikácia požiadaviek, WebGME, MVC architektúra, návrhový vzor MVC

Abstrakt v AJ

This thesis points out the possibility of facilitating and streamlining engineering work throughout the entire life cycle of a software system from its initial design through implementation to maintenance, including verification and validation of requirements for this system. It focuses on a specific area of software systems, which is the area of web applications based on MVC architecture (design of interactive applications separating business logic from output). The aim is to propose a methodology for modeling critical parts of this category of systems. Based on the proposed methodology, CASE tool will be developed by creating a DSML (domain specific modeling language) using metamodeling and extending the generic WebGME modeling platform. Finally, the methodology and CASE tool is demonstrated on an example of a real application of igc2pdf intended to generate flight documentation.

Kľúčové slova v AJ

modeling, metamodeling, requirements validation, requirements verification, WebGME, MVC architecture, MVC design pattern

57270

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Metamodelovanie, modelovanie, verifikácia a validácia požiadaviek
Requirements Metamodeling, Modeling, Verification and Validation

Študent: **Bc. František Ferenčík**
Školiteľ: **doc. Ing. Zdeněk Havlice, CSc.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:
1. Analyzovať súčasný stav v oblasti modelovania, verifikáce a validácie požiadaviek na systém.
2. Analyzovať možnosti pre metamodelovanie požiadaviek na systém.
3. Navrhnúť a implementovať metódu pre využitie modelov požiadaviek pri vývoji a údržbe systémov.
4. Na vhodnom príklade demonštrovať využitie metamodelovania, metamodelov a modelov v navrhnutej metóde.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce (hlavná časť v rozsahu 50-70 strán a prílohy, tlačená forma v nerozoberateľnej väzbe a e-forma na DVD).

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 04.05.2020
Dátum zadania diplomovej práce: 31.10.2019

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 03. mája 2020 .....

vlastnoručný podpis

Poďakovanie

Touto cestou vyslovujem poďakovanie vedúcemu mojej diplomovej práce doc. Ing. Zdeňkovi Havlicemu, CSc. za odborné vedenie, cenné rady, pripomienky a pomoc pri jej vypracovaní.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

CASE	Computer Aided Software Engineering	počítačom podporované SW inžinierstvo
CLI	Command Line Interface	Rozhranie príkazového riadku
COM	Component Object Model	model objektového komponentu
DFD	Data-flow diagram	Diagram toku údajov
DSM	Domain Specific Modelling	Doménovo špecifické modelovanie
DSML	Domain Specific Modelling Language	Jazyk doménovo špecifického modelovania
ERD	Entity Relationship Diagram	Entitno relačný diagram
GME	Generic Modeling Enviroment	prostredie generického modelovania
GUI	Graphic User Interface	grafické používateľské rozhranie
IDE	Integrated development enviroment	Integrované vývojové prostredie
JSON	JavaScript Object Notation	Objektová notácia JavaScript
MVC	Model-View-Controller architecture	Architektúra model-pohľad-kontrolór
QA	Quality Assurance	zabezpečenie kvality
RI	Reference integrity	Referenčná integrita
RTE	Round-Trip Engineering	spiatočné inžinierstvo
STD	State Transition Diagram	Diagram stavových prechodov
SQL	Structured Query Language	štruktúrovaný dopytovací jazyk
UML	Unified Modeling Language	zjednotený jazyk modelovania
SwLC	Software Lyfe Cycle	životný cyklus softvéru

Úvod

Snahou tejto práce je pomôcť softvérovým inžinierom zlepšiť a zefektívniť prácu v rôznych fázach životného cyklu softvérového systému. Súčasťou týchto fáz sú činnosti zaoberajúce sa verifikáciou a validáciou požiadaviek na systém. Príkladom je podpora spracovania požiadaviek na nový systém alebo nových požiadaviek na existujúci systém.

Pri vytváraní systémov inžinierskym prístupom využitím striktných metodík sa vytvára veľa takzvaných „throw away“ modelov a analýz, ktoré sú jednorazové a po ich použití sa už ďalej nepoužívajú z dôvodu ich komplikovanosti alebo neaktuálnosti voči živému projektu, v ktorom sa robia zmeny. Zefektívnenie prác môžeme dosiahnuť pomocou efektívneho využitia modelov takým spôsobom, kde modely budú využívané viackrát v rámci životného cyklu softvéru. Pomocou nich budeme môcť rýchlejšie odhaľovať problémové časti systému, lepšie vysvetľovať systém využitím grafickej vizualizácie, generovať kód alebo šablóny kritických častí systému.

V agilnom prístupe je situácia odlišná oproti striktným metodikám. Činnosti, ktoré priamo nevedú k výsledku alebo čiastočného doručeniu systému sa odkladajú bokom. Pokiaľ je systém úspešný a nezanikne, často vznikajú nové požiadavky na rozšírenie, vylepšenie alebo opravu. Problém nastáva po určitom čase pri potrebe implementovania zmien v systéme vyvíjanom agilným spôsobom. Napríklad chýbajúca dokumentácia dôležitých častí systému môže spôsobiť novému programátorovi veľa ťažkostí a nedorozumení. Riešením tohto problému môže byť vytvorenie lepšieho pohľadu na systém. Súčasťou systému by bol mechanizmus, CASE nástroj, ktorý pomáha sa v systéme rýchlejšie orientovať, odhaľovať interné záležitosti, ktoré by museli byť zdĺhavo skúmané v zdrojových kódoch, alebo by umožňoval tento systém meniť.

Jednodimenzionálna textová vizualizácia nemusí byť veľmi efektívna a zrozumiteľná. Vizualizácia viacdimenzionálnymi grafickými prostriedkami môže dopomôcť k rýchlejšiemu a efektívnejšiemu pochopeniu cieľového systému. Taktiež môže pomôcť v činnostiach vykonávaných v postprojektových etapách životného cyklu softvéru. Konkrétne ide o zvýšenie produktivity, zefektívnenie zaúčania, minimalizovanie nákladov na údržbu a rozširovanie systému.

1. Formulácia cieľov práce

Obsahom tejto kapitoly je stanovenie spôsobu splnenia cieľov a úloh, ktoré vyplývajú z pokynov zadania diplomovej práce. Ciele tejto práce sú zhrnuté do nasledujúcich bodov:

1. Analýza súčasných možností pre verifikáciu a validáciu požiadaviek na systém:
    - definovanie a vysvetlenie pojmov,
    - vysvetlenie ich vzájomných rozdielov.
2. Analýza súčasného stavu v oblasti modelovania a metamodelovania:
    - vysvetliť v čom spočíva modelovanie a metamodelovanie,
    - uviesť význam využitia týchto prostriedkov.
3. Analýza možností meta CASE systému WebGME:
    - predstavenie prostredia grafického generického modelovania,
    - analýza nástroja WebGME,
    - analyza jeho výhod, nevýhod a možností rozšírenia.
2. Vytvorenie DSML pre modelovanie systémov špeciálnej kategórie.
4. Vytvorenie CASE nástroja:
    - implementujúceho vytvorený DSML,
    - zameraného na podporu a vylepšenie inžinierskych prác na softvérovom systéme,
    - vytvoreného využitím metamodelovania a založeného na platforme WebGME.
5. Vytvorenie metódy pre použitie vytvoreného CASE nástroja. Bude vytvorená príručka opisujúca postupy ako správne a efektívne nástroj používať.
6. Demonštrovanie vytvoreného CASE nástroja na systéme špecifickej kategórie s názvom igc2pdf [1] vytvorenom v rámci bakalárskej práce.

7. Definícia základných pojmov

Rôzne literatúry a zdroje z oblasti softvérového inžinierstva uvádzajú nie úplne zhodné a jednoznačné definície dôležitých pojmov. Zámerom tejto kapitoly je ozrejmenie týchto pojmov.

1.1. Význam a dôležitosť softvérového inžinierstva

Softvérové projekty sa postupom času stávajú stále rozsiahlejšie a robustnejšie. Tým sa aj začala rozširovať priepasť medzi ambíciami softvérového projektu pred realizáciou a následným reálnym stavom po implementácii softvéru. Na vývoj boli vynaložené nemalé finančné prostriedky, úsilie a čas, pričom softvér bol nízkej kvality, doručený po termíne, nespĺňal sľubované vlastnosti alebo rozpočet bol značne prekročený. Na konferencii NATO Software Engineering Conference 1968 v Nemeckom meste Garmisch boli tieto problémy zlúčené a pomenované pod názvom „softvérová kríza“ [2], pričom boli navrhnuté základné postupy riešenia krízy pomocou softvérového inžinierstva. V súčasnosti aj po 50 rokoch táto téma nie je uzavretá a naďalej sa hľadajú nové a vylepšujú už existujúce postupy z dôvodu neustáleho technologického pokroku, pričom softvérové systémy sa stávajú zložitejšími a často čelia novým komplexným a rozsiahlym problémom.

Ian Sommerville vo svojej publikácii [3] softvérové inžinierstvo klasifikuje ako inžiniersku disciplínu zaoberajúcu sa všetkými aspektami, ktoré môžu nastať v procese vývoja softvéru. Je to systematický prístup k softvérovému projektu kalkulujúci s nákladmi, potrebami zákazníkov, potrebami vývojárov, harmonogramom a taktiež zodpovedá otázky spoľahlivosti.

1.2. Fázy životného cyklu softvéru

Systémoví inžinieri rozlišujú počas vývoja softvéru niekoľko odlišných fáz, ktoré na seba navzájom nadväzujú. Tieto etapy formujú počiatočné myšlienky pre vznikajúci systém cez implementáciu až po reálne používanie a udržiavanie softvéru.

Zdroje väčšinou popisujú etapy životného cyklu v závislosti na konkrétnych modeloch (vodopádový model, špirálový model a iné.). Pri zovšeobecnení projektových fáz môžeme životný cyklus rozdeliť na nasledujúce fázy znázornené v Tab. 1:

- **predprojektové** – v tejto fáze sa kladie dôraz na analyzovanie aplikačnej domény, riešeného problému a spôsoby realizácií. Taktiež sa analyzuje realizovateľnosť, vytvárajú sa finančné, časové a iné prognózy;
- **projektové** – pred implementáciou sa vykonáva návrh architektúry, komponentov a identifikujú sa kritické časti systému. Po implementácii sa vykonáva testovanie, školenia, vytváranie dokumentácie a sprístupnenie projektu reálnym používateľom;

- **postprojektové** – po nasadení nasleduje fáza údržby systému zahrňujúca napríklad testovanie a opravu nájdených chýb. Zároveň môžu vzniknúť podnety od používateľov systému pre zmenu existujúcej funkcionality alebo pridanie novej.

Predprojektové fázy	• analýza aplikačnej domény • analýza existujúcich riešení • analýza spôsobov realizácií • finančné a časové prognózy
Projektové fázy	• špecifikácia požiadaviek • definovanie ohraničenia systému • návrh architektúry a komponentov • identifikácia kritických častí • implementácia • testovanie • školenia • vytváranie dokumentácie • integrácia aplikácie do pracovného prostredia • nasadenie
Postprojektové fázy	• testovanie • oprava chýb • zmena funkcionality • pridanie funkcionality • evidovanie a vyhodnocovanie podnetov od používateľov • používanie systému

Tab. 1: Fázy životného cyklu softvérového systému [4]

Táto práca sa snaží podporiť inžinierske procesy v týchto fázach.

1.3. CASE systémy

Termín CASE označuje akýkoľvek spôsob podpory softvérového inžinierstva s využitím automatizácie. CASE môžeme chápať ako konkrétny nástroj používaný na automatizáciu jednej alebo viacerých aktivít spojených s vývojom softvéru. Medzi hlavné ciele CASE nástrojov patrí:

- zvýšenie produktivity,
- zlepšenie kvality softvéru,
- minimalizácia nákladov.

Medzi hlavné výhody CASE nástrojov patrí [5]:

- rýchlejšie dokončenie alebo doplnenie úloh,
- centralizované informácie,
- diagramy sú jednoduchšie pochopiteľné ako textový opis,
- zníženie nákladov na údržbu,
- zlepšenie kvality softvéru.

1.4. Architektúra Model-View-Controller

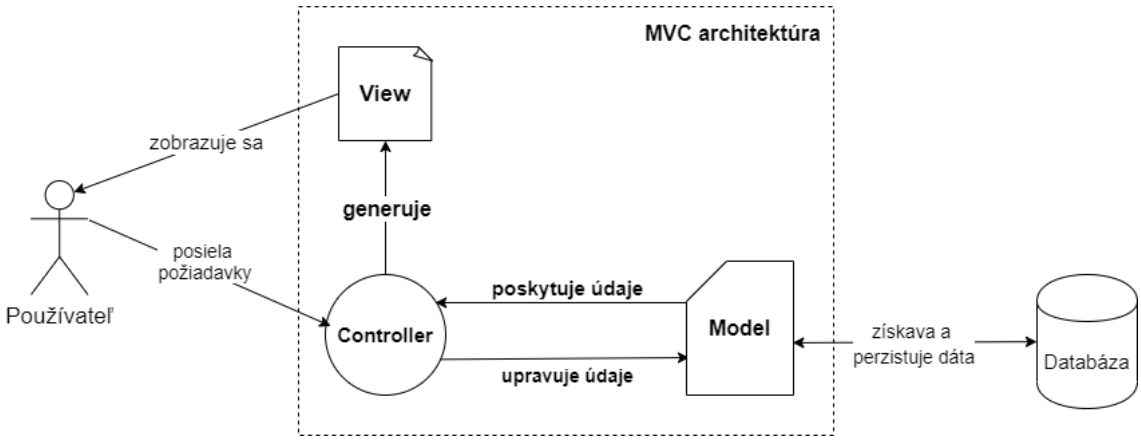
Táto práca sa zameriava na softvérové systémy postavené na MVC architektúre.

Dnešné aplikácie založené na webovej technológii si vyžadujú vysoký stupeň interakcie, ktorá sa dosahuje pomocou GUI. Cieľom je dosiahnuť čo najvyššiu použiteľnosť aplikácie, ktorá poskytuje rýchly a pohodlný prístup k službám, a z toho dôvodu musí umožňovať používateľovi sa ju rýchlo naučiť ovládať.

Pri návrhu takejto aplikácie sa kladie dôraz na vytvorenie funkčného jadra aplikácie nezávislého od jej používateľského rozhrania. Jadro interaktívnych aplikácií je založené na funkčných požiadavkách na systém a zvyčajne zostáva stabilné [17]. V praxi sa však používateľské rozhranie často mení a prispôsobuje. Je teda potrebné pri návrhu zvoliť architektúru, ktorá podporuje prispôsobenie častí používateľského rozhrania bez toho, aby spôsobili zásadné účinky na funkčnosť špecifickú pre aplikáciu alebo dátový model, z ktorého softvér pozostáva.

MVC architektúra je návrhovým vzorom poskytujúcim základnú štrukturálnu organizáciu pre interaktívne systémy. Jej komponenty sú zobrazené na Obr. 1:

- **Model** je dátová a obchodná logika. Zapuzdruje údaje dátovej vrstvy a funkcionalitu biznis logiky. Zachováva údaje aplikácie a objekty modelu načítajú a ukladajú stav modelu do databázy.
- **Pohľad (View)** tvorí používateľské rozhranie. Zobrazuje modelom spracované dáta používateľovi a tiež mu umožňuje ich modifikáciu.
- **Ovládač (Controller)** spracúva požiadavku používateľa. Používateľ v GUI vyvoláva určité URL požiadavky. Tieto požiadavky spracuje ovládač, ktorý zasiela odpoveď s vygenerovaným pohľadom obsahujúcim príslušné údaje modelu.



Obr. 1: princíp MVC architektúry

2. Verifikácia, validácia a modelovanie požiadaviek na systém

V softvérovej oblasti nie je úplne jednoznačne dohodnuté, čo verifikácia a validácia znamenajú a pokrývajú. Z tohto dôvodu táto kapitola vysvetľuje tieto pojmy.

Je dôležité, aby výsledkom vývoja systému bol produkt spĺňajúci určité vopred dohodnuté parametre. V softvérovom inžinierstve sa tieto parametre pravidelne kontrolujú pomocou zaužívaných procesov, ktoré sa tiež označujú ako kontrola kvality softvéru. Sú taktiež zahrnuté aj v SwLC.

Verifikácia a validácia sú dva úplne odlišné pojmy, ktoré ale spoločne pomenúvajú kontrolné procesy zabezpečujúce, aby softvér zodpovedal požiadavkám zákazníka.

2.1. Validácia softvéru

Úlohou validácie je nájsť odpovede vyplývajúce z otázky „Vytvárame ten správny produkt?“. Výsledkom neúspešnej validácie z používateľského pohľadu môže byť poznámka „Aplikácia nerobí to, čo som od nej očakával“.

Je to proces kontroly, či vyvíjaný softvér spĺňa potreby a očakávania koncového používateľa [6]. Cieľom je zistenie splnenia požiadaviek používateľov a kontrola, či špecifikácia bola vykonaná správne.

Štandard softvérového inžinierstva IEEE-STD-610 [15] definuje tento pojem ako proces hodnotenia systému alebo jeho komponentu počas alebo na konci procesu vývoja, aby sa určilo, či spĺňa stanovené požiadavky. Softvér sa validuje po dokončení celého systému alebo jeho komponentu či modulu. Tento proces smeruje k zisteniu a zabezpečeniu skutočnosti, aby zákazník obdržal na konci vývoja softvérový produkt, ktorý potrebuje, a či je to ten správny systém na riešenie jeho problémov. Zákazníka v podstate nemusí zaujímať, akým spôsobom sa vývojársky tím dopracoval k riešeniu, ale musí ho zaujímať, či je tento softvérový produkt preňho užitočný.

Validácia sa vykonáva prostredníctvom jednotkových, integračných, systémových a akceptačných testov.

Cieľom práce je podporiť procesy kontroly v čo najskoršej fáze, cez ktoré je možné validovať softvér nielen na konci vývoja s hotovým produktom, ale čím skôr už v prvotných fázach vývoja pomocou vhodných modelov.

2.2. Verifikácia softvéru

Verifikácia sa od validácie dosť líši a slúži na rozličné účely. Jej úlohou je zodpovedať na otázku „Vytvárame tento produkt správne ?“. Pri nesprávnej verifikácii môže vzniknúť výrok vývojára „Aplikácia nefunguje podľa schválených požiadaviek“.

Je to proces kontroly, či vyvíjaný softvér implementuje požiadavky správnym spôsobom [6].

Štandard softvérového inžinierstva IEEE-STD-610 [15] definuje tento pojem ako proces hodnotenia systému alebo jeho komponentu s cieľom určiť, či systém alebo komponent v skúmanej vývojovej fáze vyhovuje požiadavkám, ktoré boli stanovené na začiatku tejto fázy. Dôležité pre verifikáciu je to, že systém sa môže skúmať v určitej vývojovej fáze, ktorá je ešte stále vo vývoji.

Cieľom je, aby softvér zodpovedal požiadavkám na systém a návrhovým špecifikáciám.

V súčasnosti proces verifikácie zahŕňa činnosti tímu zodpovedného za QA ako sú:

- dialógy na stretnutí,
- recenzie,
- inšpekčné metódy,
- revízie kódu,
- prechod systémom (walk-through).

Významom verifikácie je identifikovať a vyriešiť aktuálne problémy a chyby v programe, ale aj chyby zdedené z predchádzajúcich fáz vývoja. V konečnom dôsledku to má významný pozitívny efekt na úsporu nákladov a času naprieč celým projektom. Je oveľa jednoduchšie a efektívnejšie odhaliť a opraviť malú chybu vytvorenú v aktuálnej fáze ako riešiť vzniknutý problém a jeho dôsledky veľmi neskoro, keď už tento problém ovplyvňuje veľké množstvo zdrojových kódov.

Cieľom práce je podporiť procesy verifikácie v čo najskoršej fáze vývoja systému.

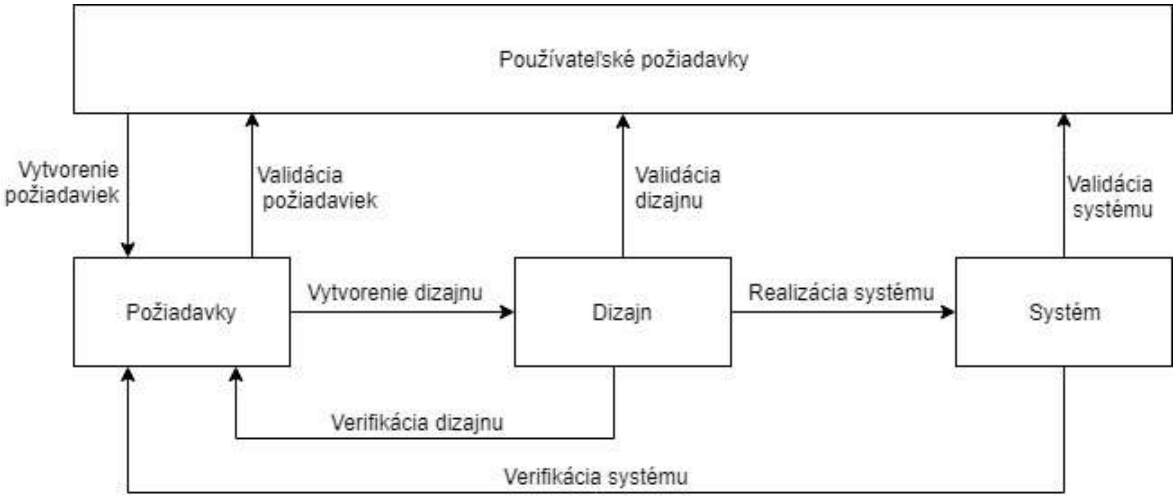
2.3. Verifikácia kontra validácia

Kombináciou verifikácie a validácie vznikajú procesy určovania, či požiadavky na systém alebo jeho komponent sú úplné a správne a systém alebo jeho komponent v každej vývojovej fáze spĺňa požiadavky alebo podmienky stanovené v predchádzajúcej fáze a konečný systém alebo komponent spĺňa stanovené požiadavky [16]. Obe metódy sú nevyhnutné a navzájom sa dopĺňajú a poskytujú rozličné postupy ako odhaliť chyby, či už na úrovni zdrojových kódov, alebo na úrovni špecifikácií systému. Taktiež sa snažia odhaliť nedostatky v implementácii v čo najskoršej fáze.

V súčasnosti sa kontrolujú pôvodné požiadavky systému, potom sa môžu vykonať revízie kódu a prechody systémom (walk-through), aby sa zúčastnené strany uistili, že vytvorené časti systému spĺňajú požiadavky klienta a tým sa zabezpečuje verifikácia systému. Po dokončení vývoja systému alebo jeho časti je potrebné sa uistiť, že tento produkt je v skutočnosti to, čo zákazník požadoval, a to je úlohou validácie.

2.4. Kedy verifikovať a validovať

Nestačí, ak sa verifikuje a validuje až finálny systém pri odovzdávaní. Je potrebné, aby boli verifikované aj požiadavky a taktiež návrh systému. Taktiež platí pravidlo, že verifikácia a validácia sa musí začať vykonávať tak skoro, ako to je možné, pretože čím skôr sa zistia rozdiely, tým menej úsilia a nákladov bude stáť nutná oprava systému.



Obr. 2: Model verifikácie a validácie systému [14]

Obr. 2 znázorňuje 3 typy validácie a 2 typy verifikácie, ktoré sú popísané nižšie [14]:

- Validácia požiadaviek – zisťuje, či sú požiadavky kompletné, jednoznačné a dôsledné a či je systém realizovateľný na základe týchto požiadaviek.
- Validácia dizajnu - overenie, či návrh aplikácie dokáže spĺňať používateľské požiadavky a zámery.
- Validácia systému – overenie, či aplikácia dokáže spĺňať používateľské požiadavky a zámery.
- Verifikácia dizajnu – proces zisťovania, či návrh systému zodpovedá požiadavkám používateľov.
- Verifikácia systému – kontrola, či systém spĺňa jeho požiadavky a či korešponduje s jeho návrhom.

2.5. Aktuálny stav verifikácie a validácie

V publikácii od Sargenta [22] sú zhrnuté aktuálne najbežnejšie používané techniky používané pri verifikácii a validácii modelov. V praxi sa tieto techniky zvyčajne kombinujú, pričom ich snahou je overiť platnosť ako čiastkových modelov, tak aj platnosť celkového modelu. Podľa môjho názoru sú najzaujímavejšie nasledujúce techniky pre statické modely, ktoré nie je možné vykonávať:

- **Inšpekcia –** organizovaný tím vývojárov, testerov a používateľov prechádzajú artefakty modelov, pričom porovnávajú konceptuálny model oproti požiadavkám.
- **Porovnávanie viacerých modelov –** model alebo výsledok sa porovnáva s iným modelom, ktorý je považovaný za validný.
- **Validácia konfrontáciou –** model je konfrontovaný s doménovým expertom a kladú sa mu otázky napríklad ohľadom správnosti konceptuálneho modelu alebo zmysluplnosti vzájomných relácií.

Niektoré z techník pre dynamické vykonateľné modely sú:

- **Animácia –** pokiaľ je model vykonateľný, tak správanie sa vykonávaného modelu je vizuálne zobrazené v závislosti od času. Pri validácii sa skúma správanie spusteného modelu a vyhodnocuje, či požadované parametre sú splnené.
- **Validita udalostí –** počet výskytov určitých udalostí simulácie modelu je porovnaný s počtom výskytu tejto udalosti v realite.
- **Test extrémnych podmienok –** model je vystavený extrémnym podmienkam na vstupe a sleduje sa, aký to bude mať vplyv na výstup. Napríklad v prípade prázdneho skladu musí byť produkcia pozastavená.
- **Validácia pomocou historických dát –** podobne ako pri validite udalostí, model len porovnáva s dáta nadobudnutými v minulosti.
- **Validácia predikciou –** opak validácie pomocou historických dát. Simuláciou sa vytvorí predikcia, ktorá sa porovná s realitou.
- **Trasovanie –** Sledovanie vykonávania špecifickej časti modelu krok po kroku pričom sa vyhodnocuje ich správnosť.
- **Turingov test –** človek, ktorý ma dobré znalosti o modelovanom systéme, je konfrontovaný s výsledkami simulácie a reálnymi dátami a skúma sa, či ich dokáže identifikovať.

2.6. Zhrnutie

Pri modelovaní požiadaviek na systém sú procesy verifikácie a validácie obzvlášť dôležité. Bez presvedčenia sa, že daný model je dôveryhodný a spĺňa požadované parametre, nie je možné ho použiť pre spoľahlivé implementovanie systému. Je to zložitý a časovo náročný proces a preto je dôležité poznať tieto odporúčania [23]:

- Verifikácia a validácia musí byť vykonávaná kontinuálne počas celého SwLC.
- Model je zostavený so špecifickým cieľom a preto musí byť posudzovaný v rámci rozsahu jeho účelu.
- Verifikácia a validácia by mala byť vykonávaná nezávislou osobou. Autor psychologicky v podvedomí zvykne označovať svoju prácu za správnu.
- Je to veľmi náročný proces vyžadujúci si veľmi dobrú znalosť modelovaného systému.
- Procesy verifikácie a validácie musia byť navrhnuté a dokumentované.
- Začať verifikovať, validovať a hľadať chyby v modeloch tak skoro, ako to je len možné.
- Validita jednotlivých modelov ešte nezaručuje validitu systému ako celku.

3. Modelovanie

Táto kapitola je venovaná modelovaniu softvérových systémov, pomocou ktorého budú predstavy dizajnérov aplikácie zhmotnené do grafickej reprezentácie jej štruktúry, vnútorného usporiadania a prepojenia jej jednotlivých častí.

3.1. Vysvetlenie a význam modelovania

Modelovanie má široké využitie naprieč rôznym inžinierskym a vedeckým odborom. V softvérovom inžinierstve je úlohou poskytnúť abstrakciu o softvérovom systéme, ktorý opisuje s potrebnou mierou informácií o detailoch. Takto vytvorené modely sú potom analyzované za účelom lepšieho pochopenia vyvíjaného systému [19]. Najlepší pohľad na systém môže poskytnúť skúmanie tohto systému z viacerých perspektív akými sú napríklad statické, dynamické modely alebo modely požiadaviek na systém.

Modelovanie teda spočíva v grafickej reprezentácii analyzovaného softvéru a viacdimenzionálna vizuálna podoba je určite názornejšia a lepšie pochopiteľná ako statická jednodimenzionálna textová forma. Taktiež zlepšuje pochopiteľnosť a komunikáciu v rámci celého tímu zapojeného do vývoja projektu.

3.1. Abstrakcia – kľúč zlepšenia produktivity vývoja

Abstrakcia je základným princípom modelovania, pomocou ktorého je možné zvýšiť produktivitu vývoja systému.

Prieskum softvérovej produktivity z roku 2005 [24] ukázal, že priemerná produktivita v jazyku Java alebo C++ (90. roky) oproti jazyku BASIC (70. roky) je maximálne iba o 20% lepšia. Podobne aj všetky ostatné súčasne používané programovacie jazyky pre všeobecné použitie neindikujú výrazný nárast produktivity v porovnaní s jazykom BASIC.

Výskum jednoznačne ukazuje, že za posledných 50 rokov (Java a C++ - momentálne najpoužívanejšie) sa produktivita programovania posunula len veľmi málo. Avšak pri pohľade do minulosti pri prechode z Assemblera na jazyk BASIC vzrástla produktivita o radikálnych 400%.

Tento obrovský 400% skok spočíva práve prechodom ku vyššej úrovni abstrakcie. Každý príkaz v jazyku vyššej úrovne akými sú Java, C++ alebo BASIC reprezentuje a je automaticky preložený do niekoľkých príkazov v jazyku Assembler. Pre príklad v praxi tak za cenu 1 riadku v Jave dostaneme 5 riadkov kódu v Assembleri.

História teda dokazuje, že zvyšovanie abstrakcie môže významným spôsobom pozitívne ovplyvniť produktivitu programovania a vývoja softvéru.

3.2. UML – jazyk unifikovaného modelovania

Súčasťou tejto práce je vytvorenie metodológie pre modelovanie systémov špeciálnej kategórie, ktorej základom sú určité modely. Tieto modely sú z veľkej miery inšpirované praxou a rokmi overenými UML modelmi, ktoré sú od 90. rokoch de facto štandardom pre modelovanie softvérových systémov.

UML – unifikovaný modelovací jazyk je viacúčelový jazyk pre modelovanie systémov grafickým spôsobom. UML modelovanie bolo navrhnuté s dôrazom na využitie nadobudnutých praktických vedomostí z oblasti návrhu a modelovania v softvérovom inžinierstve. UML poskytuje grafickú syntax pomocou ktorej je možné konštruovať modely. Teda návrhárovi je poskytnutý jazyk obsahujúci určitú slovnú zásobu spolu s jeho pravidlami, ktorých zámerom je reprezentovať systém na úrovni konceptuálnej alebo fyzickej [20]. Dôležitou vlastnosťou je znižovanie entropie systému a tým definovanie jeho štruktúry a usporiadanosti. Zatiaľ čo niektoré časti systému je možné vyjadriť jediným modelom, často vzniká potreba pre dôkladné porozumenie systému použiť viacero navzájom prepojených modelov opisujúce danú oblasť iným pohľadom.

UML neposkytuje žiadnu konkrétnu metodiku. Úlohou metodiky je poskytnúť softvérovému inžinierovi návod ako UML využiť, aké činnosti je potrebné vykonať, aké modely zostaviť aby sa zvýšila pravdepodobnosť dodania softvérového produktu v požadovanej kvalite s požadovanými parametrami. Príkladom takejto metodiky je UP (Unified Process). Taktiež UML nie je zviazané so žiadnou konkrétnou architektúrou.

Ako uvádza CTO nástroja MetaCase Dr. Steven Kelly [25], na jednej strane modelovanie pomocou UML je veľmi populárne vďaka jeho dobre navrhnutej vizuálnej reprezentácii, ktorú je možné rýchlo čítať a rýchlo získať prehľad o danom systéme. No na druhej strane veľkej časti vývojárov UML neponúka prostriedky na reprezentáciu určitých vecí v modeloch, takže potrebujú UML o niečo rozšíriť. Ďalšia časť vývojárov považuje UML za zbytočne komplikovaný a chceli by ho zredukovať na základné elementy. UML sa snaží byť nástrojom pre všetko a pre všetkých, čo je pri dnešnej komplexnosti systémov veľmi ťažká úloha. Je priam až nemožné spojiť protichodné požiadavky spomenutých 2 skupín vývojárov. Tento problém má za následok to, že nie je možné zvyšovať abstrakciu.

3.3. DSM – doménovo špecifické modelovanie

Doménovo špecifické modelovanie je jedna z metodík v softvérovom inžinierstve používaná na vývoj softvéru. DML sa špecifikuje na cieľovú doménu softvérového projektu a jeho hlavný význam je v tom, že určité úlohy a problémy rieši oveľa lepšie a efektívnejšie ako univerzálne modelovacie techniky. Zameriava sa na 2 základné koncepty [21], ktorými sú:

- **Vyššia úroveň abstrakcie** – DSML popisujú systém na vyššej úrovni abstrakcie oproti univerzálnym modelovacím jazykom.
- **Automatizácia pomocou generátorov –** DSM sa snaží v čo najväčšej miere využiť namodelované modely pre generovanie zdrojových kódov. Zameriava sa na kritické časti systémov a poskytuje oproti UML väčšie možnosti v tomto smere. Taktiež je špecializovaný na konkrétnu cieľovú platformu, na ktorej bude bežať vyvíjaný systém.

DSM vyžaduje navrhnutie a zadefinovanie jazyka, ktorý sa označuje aj ako metamodel. Táto práca definuje jazyk pomocou metamodelov v kapitole 7.

DSML vďaka svojej snahe o kompletnosť je veľmi vhodný na generovanie cieľového zdrojového kódu [25]. Táto metóda vedie k prístupu, v ktorom tvoria jadro úsilia vývoja softvéru grafické modely, z ktorých je možný prechod k implementácii pomocou generátorov zdrojových kódov a taktiež je aj výborným zdrojom pre dokumentáciu. Táto dokumentácia obsahuje vizuálny pohľad na celý systém na vysokej úrovni abstrakcie, čo môže výrazne dopomôcť k včasnej a kvalitnejšej verifikácii a validácii.

3.4. Zhrnutie – UML kontra DSML

Predstavili sme si UML, ktorý má podľa môjho názoru veľmi podstatnú vlastnosť, ktorú treba pred jeho použitím poznať. Je ňou flexibilita použitia pre širokú oblasť uplatnenia. Prečo to môže byť problém? Pretože neumožňuje zvýšiť úroveň abstrakcie nad najnižší spoločný menovateľ všetkých používateľov, ktorým chce vyhovieť.

Programátori sa vždy snažia zvyšovať produktivitu zlepšovaním abstrakcie, vizualizácie a automatizácie. DSM tieto koncepty považuje ako svoje základné piliere. Výsledok použitia DSM je vyššia úroveň abstrakcie ako UML, pretože používa priamo jazyk aplikačnej domény. Takýto jazyk je veľmi silný avšak výrazne obmedzený na použitie iba pre špecifickú kategóriu systémov, pre ktorú bol vyvinutý.

4. Metamodelovanie

Keďže budeme metamodelovanie využívať vo vytváranom CASE nástroji, je vhodné si tento dôležitý pojem priblížiť, čo bude obsahom tejto kapitoly.

Pojem metamodelovanie je proces vytvárania metamodelov. Metamodel sa odvíja od slova model, ktoré sa používa vo viacerých vedných odboroch. Väčšinou však ide o reprezentovanie nejakého objektu, ktorý odráža jeho obraz z reálneho sveta. Taktiež sa model označuje ako zjednodušenie reality.

Slovo „meta“ sa používa pri pokuse o zvýšenie úrovne abstrakcie významu daného pojmu. Napríklad pojem metadáta popisuje, že sa jedná o nejakých dátach, ktoré popisujú iné dáta. Reálnym príkladom toho je nejaký súbor uložený v súborovom systéme operačného systému. Pojem dáta označuje v tomto kontexte obsah daného súboru a pojem metadáta označuje dáta súvisiace s týmto súborom, ktoré ho opisujú. Metadáta by boli v tomto prípade formát súboru, dátum vytvorenia a úpravy súboru, vlastník súboru, prístupové práva alebo veľkosť súboru.

Metamodelovanie môžeme taktiež analogicky interpretovať ako modelovanie modelovania [7], respektíve metamodel je model popisujúci vlastnosti a koncepcie iného modelu.

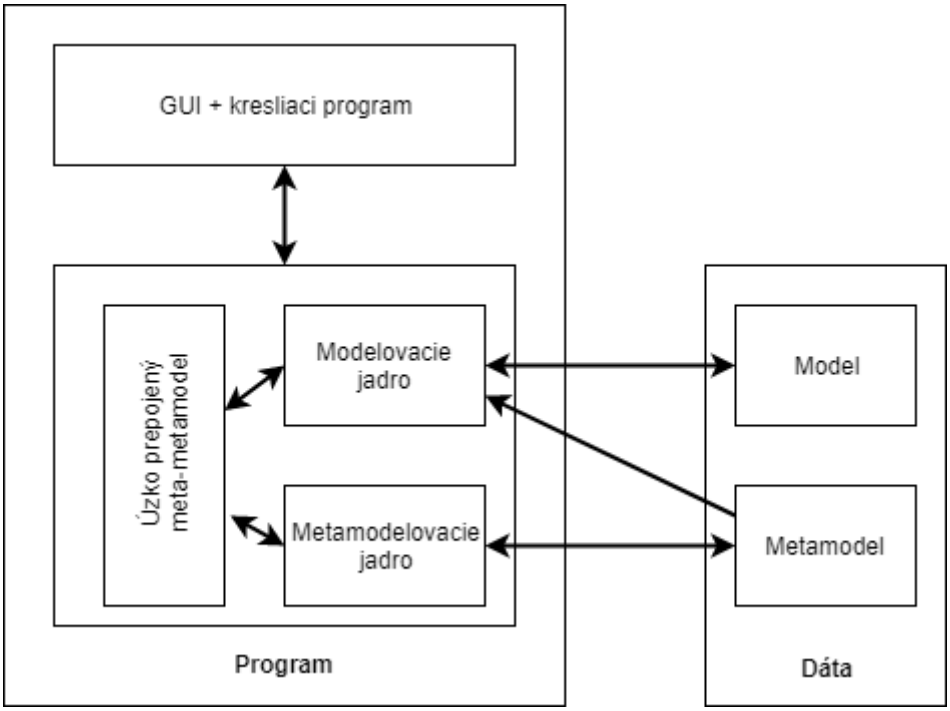
4.1. MetaCASE

Pri modelovaní sa využívajú CASE nástroje a pomocou metamodelovania sa tieto nástroje popisujú pomocou MetaCASE nástrojov. Označujú sa niekedy aj ako softvér riadený metamodelom [7], pretože zmenou metamodelu sa môže zmeniť chovanie programu.

Na Obr. 3 je znázornená schéma MetaCASE nástroja, ktorého hlavné prvky sú [7]:

- **GUI** – poskytuje rozhranie pre interakciu nástroja s používateľom. Poskytuje taktiež vhodné kresliace nástroje;
- **Úzko prepojený meta-metamodel** – interpreter metamodelovacieho jazyka;
- **Modelovacie jadro** – umožňuje modelovanie modelov;
- **Metamodelovacie jadro** – jeho prostredníctvom je naprogramovaný metamodel v jazyku definovanom v metametamodeli. CASE a MetaCASE nástroje sa odlišujú hlavne v tejto časti. Táto časť umožňuje naprogramovanie vlastnej metodológie;
- **Model** – údaje reprezentujúce model perzistované napríklad v databáze;

- **Metamodel** – údaje reprezentujúce metamodel môžu byť perzistované v databáze.



Obr. 3: Schéma MetaCASE nástroja [7]

4.2. Výhody metamodelovania

Najčastejším dôvodom vývoja metaCASE nástrojov je ten, že vývojári CASE nástrojov vyvíjajú k nemu metaCASE nástroj, ktorý neskôr transformujú do CASE nástroja bezprostredne pred jeho vydaním. Tento postup im zvyčajne zaisťuje vyššie čísla predajov nových verzií. To je dosiahnuté rýchlou a lacnou rozšíriteľnosťou CASE nástroja, takže inovácia softvéru je tým pre vývojový tím omnoho jednoduchšia, pretože požadované zmeny nie je nutné robiť v zdrojových kódoch, ale stačí vykonať zmeny len v metamodeli, ktorý zmeny transformuje do zdrojových kódov automaticky a nová verzia je pripravená na zverejnenie.

Hlavné výhody využitia metamodelovania môžeme zhrnúť nasledovne [8]:

- Rýchlejší vývoj nových CASE nástrojov;
- Nižšie zriaďovacie náklady;
- Podpora ľubovoľného modelu;
- Tvorba nových metamodelov a modelov;
- Možnosť upravovať už vytvorené metamodely a modely;
- Generovanie zdrojových kódov;
- Možnosť výmeny a zdieľania metadát medzi metamodelmi;
- Možnosť doplňovať metamodely o nové metadáta.

4.3. Zhodnotenie – využitie metamodelovania pre modelovanie systémových požiadaviek

V kapitole 3.4 boli konfrontované metódy UML a DSM s takým záverom, že vďaka svojmu princípu riešenia problémov cieľového systému mechanizmom ušitým na mieru DSM dokáže byť radikálne efektívnejší v procesoch v rámci životného cyklu programu.

Praktickým výstupom práce je vytvorenie CASE nástroja zameraného na modelovanie webových aplikácií založených na návrhovom vzore MVC. Tento CASE nástroj bude implementovať práve DSML navrhnutý špeciálne pre špecifikovanú aplikačnú doménu. Bude vytvorený rozšírením generického modelovacieho prostredia WebGME. Základom tohto prostredia je implementácia princípov metamodelovania a práve pomocou nich je možné vytvoriť jazyk DSML obsahujúci určité objekty mapujúce dôležité aspekty cieľového systému spolu s pravidlami, ako ich používať a ako ich vzájomne prepájať. Teda vhodným vytvorením a nakonfigurovaním metamodelov určíme spôsob, akým sa budú zostavovať modely nového CASE nástroja a teda akým spôsobom sa bude vyvíjať aplikácia z tejto aplikačnej domény.

Na úrovni metamodelov je možné nakonfigurovať rôzne obmedzenia, ktoré pomáhajú zabezpečiť integritu vyvíjaného systému a tak dopomôcť zlepšeniu procesov verifikácie a validácie. Napríklad taký proces v diagrame toku údajov musí obsahovať minimálne jeden vstup a jeden výstup. Ak v ňom absentuje vstup, tak ho môžeme označiť ako tzv. zázračná skrinka produkujúca výsledok na základe ničoho, alebo ak v ňom absentuje výstup, tak je to čierna skrinka v ktorej sa informácia stráca. Obmedzenie nakonfigurované pre riešenie tohto problému teda upozorní vývojára na nekonzistentnosť v dizajne už pri jeho vytváraní a prípadná chyba takéhoto charakteru je odhalená instantne už vo fáze návrhu, kedy jej odhalenie a odstránenie má najmenší finančný a časový dopad.

5. Metamodelovací nástroj WebGME

Obsahom tejto kapitoly je predstavenie open-source nástroja WebGME, ktorý je použitý pre vytvorenie CASE nástroja. Ako už názov naznačuje, WebGME je škálovateľná a modifikovateľná aplikácia založená na webovej platforme. Táto platforma novej generácie poskytuje metamodelovacie prostredie umožňujúce spoluprácu viacerých vývojárov. Je určená pre budovanie nových nástrojov založených na modeloch.

5.1. Prostredie grafického a generického modelovania

Prostredie grafického modelovania je určené na podporu vývoja systémov. Obsahuje sadu nástrojov pre modelovanie, analýzu modelov, generovanie kódu a simuláciu. Tieto nástroje sú určené pre podporu procesov sprevádzajúcich návrh softvérových systémov. Systémové špecifikácie sú reprezentované doménovými modelmi, ktoré podporujú činnosti súvisiace s návrhom softvéru využitím automatizovaných systémových analýz, generovaním zdrojových kódov, simuláciami alebo automatickom konfigurovaní a integrovaní jednotlivých komponentov navrhovaného systému.

Veľkú prekážku brániacu jeho širokému uplatneniu predstavujú pomerne vysoké zriaďovacie náklady. Dôsledkom toho tieto nástroje sa využívajú hlavne v rentabilných doménach s veľkým trhom, ktorý kompenzuje vysoké zriaďovacie náklady.

Riešením tohto problému je zaobstaranie ľahko konfigurovateľného generického nástroja, ktorý bude poskytovať všeobecné funkcie grafického vývojového prostredia. Týmito funkciami sú napríklad vytváranie a spravovanie projektov, vytváranie upravovanie diagramov a iné. Zároveň tento nástroj musí umožňovať jednoduché prispôsobenie pre použitie v cieľovej doméne. Platí však pravidlo, že úsilie a náklady spojené s prispôsobovaním nástroja musia byť menšie ako vývoj novej sady nástrojov špeciálne pre cieľovú špecifickú doménu [12]. V praxi to znamená, že vývoj plne funkčného modelovacieho prostredia s využitím spomínanej generickej konfigurovateľnej sady nástrojov môže trvať rádovo od niekoľkých hodín v závislosti od náročnosti cieľovej modelovacej metodológie. No v prípade vývoja vlastného prostredia na mieru sa vývoj zvykne pohybovať v rádoch stoviek až tisícok človeko-hodín.

Jedným z vyššie opísaných grafických generických konfigurovateľných modelovacích nástrojov je WebGME, ktorému sa budeme venovať v nasledujúcich kapitolách.

5.2. Predstavenie nástroja WebGME

Architektúrou aplikácie WebGME [13] je model klient server, pričom serverovú časť zabezpečuje technológia NodeJS, ktorej úlohou je primárne ukladať a poskytovať potrebné základné

neupravené a nespracované dáta modelov, ktoré ďalej distribuuje všetkým klientom. Serverová časť je prepojená s klientskou časťou prostredníctvom webového prehliadača, ktorý nesie veľkú časť výpočtového výkonu a pracovného zaťaženia.

Aplikácia vo webovom prehliadači beží ako tenký klient, čo je dôležité obzvlášť pri veľkých modeloch, pretože sa nenačítava celý model, ale len jeho požadované oblasti.

Práca s modelmi v tomto nástroji je uložená v centralizovanom systéme GIT, ktorý je riadený verziami. GIT zaručuje, že všetky vykonané zmeny v modeloch sú navzájom nezávislé. Vďaka tomu je možné sa v priebehu práce kedykoľvek vrátiť k ľubovoľnému predchádzajúcemu stavu.

Webový prehliadač, klientska časť implementuje behaviorálny návrhový vzor „Observer“ [9], v preklade pozorovateľ, čím nezavádza celý model, ale prihlási sa na odber zmien a počúva na udalosti v jednotlivých častiach v rámci modelovej hierarchie. Komunikácia so serverovou časťou spočíva hlavne v získavaní surových nespracovaných dát popisujúcich modely. V prípade viacerých verzií tých istých dát sa konflikty medzi nimi riešia využitím metódy tzv. „diff“ v preklade rozdiel. Práca v nástroji je teda rozdelená do malých potvrdení, ktoré vytvárajú v systéme GIT samostatný záznam, ktorý si všetci práve kolaborujúci klienti môžu získať a zlúčiť so svojou verziou modelov, čo zaručuje, že všetci klienti môžu mať tú istú verziu, teda ten istý stav všetkých modelov.

WebGME bol navrhnutý s dôrazom na rozšíriteľnosť nástroja. Poskytuje možnosti pre vytvorenie doplnkov pomocou jazyka JavaScript alebo Python. Tieto doplnky môžu byť vykonávané ako na serverovej časti tak aj v prehliadači v klientskej časti alebo spoločne v oboch častiach.

5.3. Rozšíriteľnosť a prispôsobiteľnosť WebGME

WebGME poskytuje rôzne možnosti prispôsobenia a rozširovania aplikácie [27]:

- **RESTful API** – umožňuje rozširovať aplikáciu ľubovoľným programovacím jazykom.
- **Vlastné cesty pre REST API** – umožňujú využiť prístup k úložiskovému alebo autentifikačnému API.
- **Doplnok** (plug-in) – vlastné rozšírenie WebGME. Jeho vykonávanie je inicializované nejakou akciou alebo priamo používateľom. Sú určené pre vytváranie, dopytovanie alebo interpretáciu modelov (napr. generovanie kódu). Umožňujú vykonávanie rovnakého kódu na strane servera aj prehliadača.
- **Vykonávateľ** (executor) – mechanizmus middlewaru umožňujúci externým službám registrovať a prijímať úlohy. V praxi ide napríklad o prípad, že nejaký doplnok generuje spustiteľný kód z modelov a namiesto manuálneho preberania súborov a zadávania príkazov doplnok kód vykoná a vráti výsledok späť modelu.

- **Obmedzenia** (constraints) – umožňuje pridať vlastné obmedzenia. Napríklad na zabezpečenie toho, aby mená modelov obsahovali iba alfanumerické znaky.
- **Rozšírenie** (addon) – na rozdiel od doplnkov sa vykonávajú nepretržite a počúva na registrované udalosti projektu. Taktiež môžu využiť vrstvu jadra aplikácie. Napríklad periodicky vyhodnocujú vyššie spomínané obmedzenie.
- **Webhook –** umožňuje vyvolanie udalosti na základe zmien v projekte, modeli alebo úložisku. Napríklad vykonanie *commitu*, vytvorenie novej vetvy alebo tagu.
- **Rozloženie** (layout) – slúži na konfiguráciu používateľského rozhrania.
- **Vizualizér** (visualizer) – umožňuje pridanie vlastného komponentu do všeobecného GUI.
- **Dekoratér** (decorator) – jeho úlohou je zabezpečiť špecifické vykreslenie konkrétneho uzla modelu v editore nástroja.

Pre účely vytvorenia CASE nástroja táto práca bude využívať hlavne doplnky (automatizované vytváranie artefaktov) a dekoratéry (doménovo špecifická vizualizácia modelu).

5.4. Výhody WebGME

Obsahom tejto podkapitoly je zhrnutie najvýznamnejších výhod nástroja WebGME [10]:

- **Prototypická dedičnosť –** Modely vo WebGME sú prototypy. Z každého prototypu môže byť vytvorená inštancia, ktorá je kópiou modelu. Medzi inštanciou a pôvodným modelom je vytvorená závislosť, čo spôsobuje, že zmeny vykonané v rodičovskom modeli sú ihneď propagované aj do inštancií potomkov tohto modelu.
- **Riadenie verzií -** Používateľ si nemusí vždy sťahovať a aktualizovať celú databázu projektu, ale len jeho nevyhnutné časti , na ktorých práve pracuje.
- **Online kolaborácia –** Veľmi dobre zvládnutá spolupráca viacerých členov vývojového tímu na tom istom projekte. Zmeny vyvolané používateľom sú automaticky propagované ostatným používateľom. Vývojári môžu pracovať taktiež na vlastných paralelných vetvách toho istého projektu bez toho, aby ovplyvnili iné vetvy a používateľov. Výhodou je taktiež jednoduchá možnosť vrátenia sa späť k staršej verzii, čo je užitočné najmä v prípade výskytu chyby, alebo potrebe sa vrátiť na predchádzajúcu verziu.
- **Okamžitá propagácia zmien –** Väčšina iných modelovacích nástrojov v prípade zmien v modeloch alebo metamodeloch potrebuje manuálnu kompiláciu.
- **Bez striktného rozdelenia –** Metamodel nie je úplne oddelený od modelu využitím dedičnosti a teda neexistuje striktne ohraničenie medzi samotným nástrojom a meta časťou, čo umožňuje rýchle propagovanie zmien z metamodelu.

- **Prístupnosť databázy –** Ako už bolo spomenuté, prehliadač si sťahuje z databázy len časti modelov, na ktorých pracuje, čím znižuje pamäťovú náročnosť a zároveň umožňuje pristupovať ku všetkým častiam databázy, aj keď je veľmi rozsiahla.
- **Práca v režime offline –** Po stiahnutí si potrebných údajov z databázy do prehliadača je možné pracovať bez internetového pripojenia. Pri pripojení do siete sú následne tieto zmeny synchronizované s hlavným úložiskom.
- **Platformová nezávislosť –** Zjednodušenie inštalačného a aktualizačného procesu.

5.5. Nevýhody WebGME

Tento nástroj je ešte stále vo vývoji a má priestor na zlepšenie. Niektoré z týchto aspektov sú:

- Prepojenia medzi artefaktmi modelu sú vizualizované šípkami, ktoré systém často nie úplne optimálne vizualizuje. V prípade viacerých prepojení toho istého artefaktu sa tieto šípky zlievajú a smer spojenia s popisom sú ťažko čitateľné.
- Inštancia artefaktu z jedného modelu nemôže byť použitá v inom modeli. Pri pokuse o takéto použitie sa vytvára nová inštancia dediaca daný artefakt. Obídením tohto obmedzenia sme museli využiť vizualizér *Crosscut,* čo nie je optimálne riešenie vyúsťujúce v poklesom produktivity.
- Vizualizér *Crosscut* obsahuje často vyskytujúcu sa chybu, pri ktorej po načítaní tohto pohľadu model neobsahuje všetky artefakty, alebo nie sú kompletné. Taktiež prepojenia artefaktov nevykresľuje ako šípky, ale ako základné objekty. Pri prepnutí sa do iného vizualizéra a potom naspäť sa zvyčajne tento problém odstráni.
- Používateľ musí namodelovať všetko na jednej predvolenej stránke, ak chce vytvoriť viac modelov [10]. Musí byť možné použiť každý stavebný prvok ako stavebný prvok pre iný jazyk. Preto musí používateľ z hierarchie kompozície alebo zo záložiek otvoriť modelový prvok, pričom sa otvorí nová stránka, na ktorej nie je modelovanie možné, nakoľko po otvorení všetky elementy zmiznú.
- Nie je možné jednoducho pridať nad triedu pre objekty, pretože vzťah dedičnosti je silne viazaný. Tým nie je možné v strede dedičnej hierarchie pridať nový prvok alebo odstrániť existujúci.
- Chýbajúca podpora spájania vetiev [11]. Ak by bol taký mechanizmus implementovaný, bolo by možné rozšíriť režim kolaborácie aj na iné prípady ako sú súbežné aktualizácie modelov.

5.6. Zhodnotenie

Platforma založená na princípe generického modelovania WebGME ponúka vysokú mieru flexibility pre riešenie širokého spektra problémov, pretože poskytuje možnosť pomocou metamodelovania zadefinovať vlastné metamodely, ktorými je možné zadefinovať prvky modelovacieho jazyka a pravidlá, akými tento jazyk môže byť použitý.

WebGME pre účely tejto práce umožňuje implementáciu metódy DSM tým, že pomocou metamodelov je možné vytvoriť kompletný DSML. Po zadefinovaní metamodelov tohto DSML je možné hneď začať pracovať a modelovať v tomto novo vytvorenom jazyku.

Taktiež platforma WebGME prichádza s veľkým množstvom funkcií od správy projektov, cez možnosti modifikovateľnosti tohto nástroja, až po nástroje podporujúce online kolaboráciu. Tieto všetky dôležité funkcie teda firma, ktorá sa rozhodne vytvoriť si vlastný CASE nástroj na to, čo potrebuje, nemusí programovať.

Využitie WebGME má pozitívny vplyv na časovú aj finančnú rentabilitu zriadenia nového CASE nástroja založeného na platforme WebGME oproti vývoju úplne nového riešenia. Vývoj riešenia na platforme WebGME môže trvať rádovo v horizonte niekoľkých desiatkach človeko-hodín v závislosti od komplexnosti. Na druhej strane pri vývoji úplne nového vlastného riešenia sa môžeme pohybovať rádovo v stovkách až tisíckach človeko-hodín.

Vlastné CASE nástroje na mieru si zvyčajne môžu dovoliť financovať len veľké firmy a teda táto platforma dáva možnosť aj stredným a malým firmám si vytvoriť svoj vlastný CASE nástroj pre riešenie svojich špecifických problémov.

6. Vytvorenie DSML a metodológie modelovania systémov

Obsahom tejto kapitoly je navrhnutie metodológie modelovania systémov, ktorá bude následne implementovaná do nástroja CASE. Pre názornejšiu ilustráciu modelov bude použitá jednoduchá aplikácia internetového obchodu.

6.1. Podporované triedy programov

Cieľom nie je vytvoriť univerzálny nástroj na riešenie komplexných problémov, tie sú väčšinou veľmi rozsiahle, ťažkopádne a nie optimálne pre riešenie konkrétnych špecifických problémov.

Úžitok by mal mať taký, že nebude podporovať len jeden konkrétny projekt, ale triedy nejakých podobných projektov. Touto triedou sú webové aplikácie založené na návrhovom vzore MVC. Nástroj svojimi modelmi podporí najdôležitejšie kritické časti takejto aplikácie.

Súčasným trendom pri vývoji webových systémov je použitie zaužívanej MVC architektúry popísanej v kapitole 1.4. To bol aj hlavný dôvod zamerania sa práve na aplikácie implementujúce práve tento návrhový vzor.

6.2. Kritické časti systému

Počas návrhu metodológie pre modelovanie určitej kategórie systémov je potrebné identifikovať kritické časti, ktoré musia byť modelmi podporované. Keď sa zameriame na architektúru MVC, tak z jej základných konceptov sú badateľné kritické časti takejto aplikácie.

Na základe mojich praktických skúsenosti s vývojom systémov tejto kategórie som identifikoval tieto kritické časti:

- dátová vrstva,
- grafické používateľské rozhranie,
- riadenie dátového toku,
- riadenie stavov.

6.3. Modelovanie dátovej vrstvy – Model diagram

Dáta sú základným stavebným prvkom každého systému a preto je nutné im venovať patričnú pozornosť. V prípade webovej aplikácie je potrebné namodelovať dátové entity perzistované v databázovej vrstve a vzťahy medzi jednotlivými dátovými entitami. Inšpiráciou pre modelový diagram je fyzický entitno-relačný diagram, tiež známy ako ERD, ER Diagram alebo ER model. Je to typ štrukturálneho diagramu používaný pri návrhu databázy. ERD obsahuje rôzne symboly a konektory, ktoré vizualizujú hlavné entity systému a vzájomné vzťahy medzi týmito entitami.

Modelový diagram obsahuje:

- **Entita -** reprezentuje databázovú tabuľku. Je to definovateľná vec alebo pojem v systéme. Napríklad používateľ, objednávka, produkt;
- **Atribúty entity -** reprezentuje stĺpec tabuľky. Charakterizuje druh informácie;
- **Index -** databázová štruktúra vylepšujúca operácie získavania indexovaných dát;
- **Primárny kľúč -** špeciálny typ atribútu. Jednoznačne identifikuje záznam v tabuľke;
- **Cudzí kľúč -** referencia na primárny kľúč. Identifikuje vzťahy medzi entitami;
- **Vzťah -** definuje vzájomný súvis dvoch entít;
- **Kardinalita -** vzťahy sú definované počtom možných výskytov entít vo vzájomnom vzťahu;
- **Vzťahové manipulačné obmedzenia -** rozšírenie oproti ERD. Spôsob zabezpečenia RI.

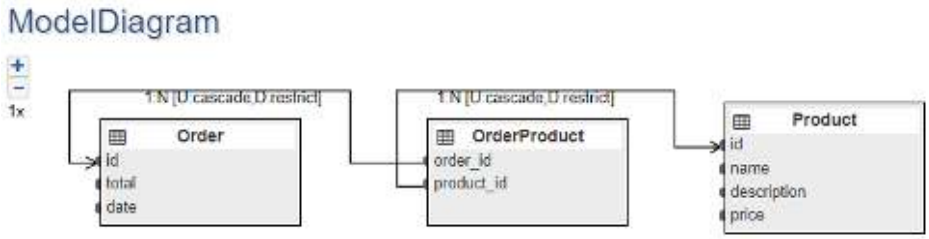
Klasický ERD diagram bol rozšírený podľa vzoru v literatúre [18] o možnosť definovať pravidlá referenčnej integrity RI pre konkrétny vzťah. Tieto pravidlá RI je možné definovať pre operácie:

- **U - update** - aktualizácie záznamu;
- **D – delete** - vymazanie záznamu.

Spôsoby, akými je možné zabezpečiť RI pre spomenuté pravidlá, sú tieto:

- **restrict** – reštrikčný - operácia je zakázaná;
- **set null** – nulitný - operácia je povolená a RI je zabezpečená nastavením hodnoty *null* na odkazované záznamy;
- **cascade** – kaskádový – operácia je povolená a RI je zabezpečená opakovaným použitím operácie pre odkazované záznamy.

Na Obr. 4 je zobrazený vzorový *ModelDiagram* pre definovanie modelu. Vidíme na ňom 3 navzájom prepojené entity s definovanými atribútmi, typmi prepojenia a pravidlami zabezpečenia RI. Ide o jednoduchú schému prepojenia objednávok a produktov cez prepojovaciu tabuľku, ktorá zabezpečuje, že jedna objednávka môže obsahovať viacero produktov a zároveň jeden produkt môže byť priradený do viacerých objednávok.



Obr. 4: Vzorový *ModelDiagram* databázovej vrstvy

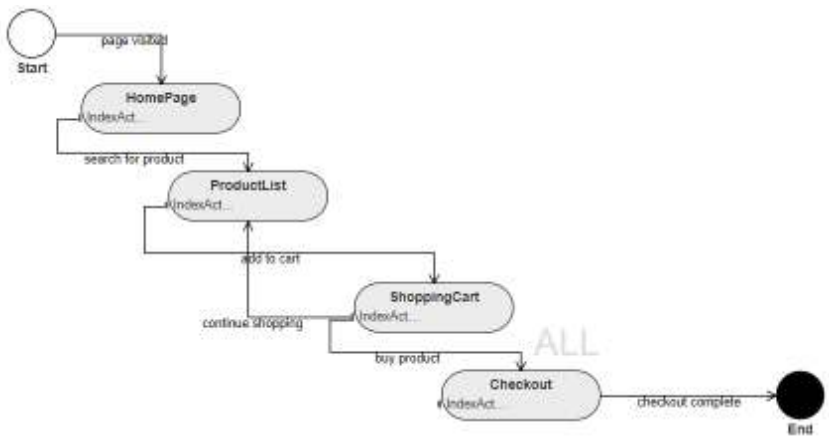
6.4. Modelovanie riadenia stavov – *ControllerDiagram*

Používateľ webovej aplikácie sa počas práce prepína medzi rôznymi obrazovkami, na ktorých vykonáva určité akcie. Množinu týchto obrazoviek a prechodov môžeme vnímať aj ako stavy aplikácie a prechody medzi nimi. Teda inšpiráciou pre *ControllerDiagram* je STD diagram stavových prechodov.

ControllerDiagram obsahuje nasledujúce komponenty:

- **Akcia** – analógia stavu v STD. Spracovanie URL požiadavky vyvolá spustenie konkrétnej akcie a teda používateľ pri načítaní URL adresy sa dostane do konkrétneho stavu v rámci aplikácie. Napríklad načítanie domovskej stránky.
- **Terminátor štart** – iniciálny stav aplikácie.
- **Terminátor koniec** – koncový stav aplikácie.
- **Kontrolér** – združuje viaceré akcie do jedného významového celku. Napríklad akcie súvisiace so zobrazením produktu v internetovom obchode.
- **Prechod** – orientované spojenie predchádzajúcich komponentov určujúce nasledujúci stav. Prechodu je možné pridať informáciu, ktorá obsahuje podmienku vyvolania prechodu. Napríklad po autentifikovaní je používateľ presmerovaný do zoznamu jeho objednávok. Pokiaľ prechod nesmeruje ku konkrétnej akcii ale iba na kontrolér, tak automaticky sa za cieľ prechodu považuje akcia označená ako index.

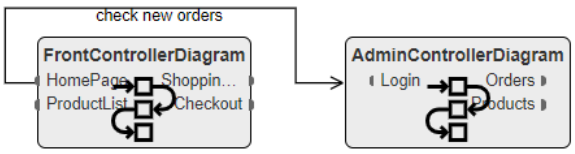
Na Obr. 5 je namodelovaný vzorový diagram postupnosti stavov jednoduchého internetového obchodu. Po príchode na stránku je používateľ na domovskej stránke, pričom po použití vyhľadávania sa dostane na obrazovku so zoznamom produktov, z ktorej si môže pridať produkt do košíka. V tomto bode sa môže vrátiť späť na zoznam produktov alebo pokračovať k vybaveniu objednávky. Po jej vybavení ukončuje prácu v aplikácii.



Obr. 5: Controller diagram – vzorový stavový diagram

Diagramy typu *ControllerDiagram* môžu združovať *Controller* v rámci väčších logických celkov – modulov. Na

Obr. 6 je namodelované prepojenie modulov aplikácie. Modul *Front* obsahuje primárne služby biznis logiky internetového obchodu a modul *Admin* obsahuje služby súvisiace s vybavovaním objednávok a správou produktov.



Obr. 6: Kompozícia modulov pomocou *ControllerDiagram*

6.5. Modelovanie používateľského rozhrania – View diagram

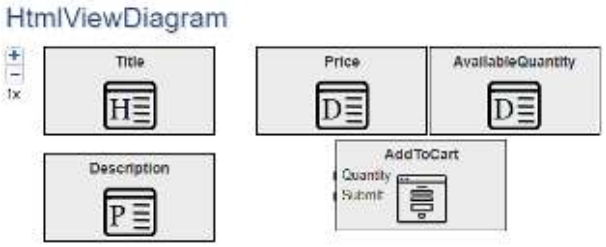
Ďalšou kritickou časťou webovej aplikácie je rozhranie GUI. Budeme ho taktiež modelovať pomocou *View* diagramu.

Tento diagram sa nevenuje čisto len reprezentácii GUI. Zvyčajne po spracovaní požiadavky je potrebné vrátiť nejaké dáta, ktoré môžu byť v rôznych formátoch. Najpoužívanejší je formát odpovede v HTML kóde, ktorý následne prehliadač interpretuje do grafickej podoby vo forme GUI. Taktiež existujú ďalšie formáty na prenos dát, ktoré sa používajú na komunikáciu s inými programami. Je to napríklad formát JSON. Taktiež odpoveďou na požiadavku môže byť aj napríklad PDF súbor.

Diagram *View* obsahuje nasledujúce komponenty:

- **Formát** – definícia druhu odpovede – HTML, JSON, File a iné.
- **HTML element** – základná jednotka obsahujúca nejakú informáciu alebo umožňujúca zápis vstupu od používateľa. Elementy tvoria výsledný HTML dokument tvoriaci GUI. Napríklad nadpis, formulár, textový vstup, odkaz, zaškrtávacie pole atď.
- **JsonObject** – používa sa v prípade JSON formátu odpovede. Obsahuje dáta v štruktúrovanej textovej podobe typu kľúč - hodnota. Napríklad informácia o výsledku dopytu množstva produktu na sklade.
- **FileStream** – odpoveďou je súbor v určitom formáte. Napríklad PDF súbor.

Na Obr. 7 je namodelovaná obrazovka stránky produktu na jednoduchom internetovom obchode. Stránka obsahuje názov, popis, cenu produktu, dostupné množstvo na sklade a formulár pre pridanie do košíka obsahujúci ďalší model formulára obsahujúci pole pre zadanie množstva a tlačidlo na pridanie do košíka.



Obr. 7: *ViewDiagram* – GUI obrazovka stránky produktu

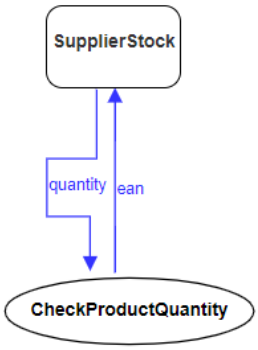
6.6. Riadenie dátového toku – DataFlow diagram

Aby bolo jasné odkiaľ sú spracovávané dáta a kam sa ukladajú alebo kam sa zobrazujú výsledky operácií, je potrebné tento proces taktiež namodelovať v *DataFlow* diagrame. Jeho inšpiráciou je DFD – diagram toku údajov.

DataFlow diagram obsahuje nasledujúce komponenty:

- **Proces** – akýkoľvek proces, ktorý na základe vstupov mení dáta a produkuje výstup. Napríklad proces vytvorenia objednávky s odoslaním notifikačných emailov.
- **Vstup** – vstupné údaje pre proces. Napríklad emailová adresa pre notifikačnú správu.
- **Výstup** – výstupné dáta procesu. Napríklad perzistovanie novej objednávky v databáze.
- **Externá entita** – systém tretej strany, ktorý posiela alebo prijíma dáta. Napríklad server, ktorý na vyžiadanie poskytne množstvo produktov na sklade dodávateľa.
- **Dátové úložisko** – súbory alebo databázové entity udržiavajúce informácie.
- **Tok** – znázorňuje smer toku informácie. Taktiež obsahuje aj textový popis, ktorý bližšie určuje charakter prenášanej informácie.

Na Obr. 8 sa nachádza redukovaný *DataFlow* diagram pre kontrolu dostupného množstva produktu na sklade dodávateľa cez jeho API. Diagram je v tomto kroku nekompletný. Skutočný význam dostane až pri jeho **prepojení** s ostatnými diagramami a to je obsahom kapitoly 6.7.2.



Obr. 8: Redukovaný *DataFlow* diagram zistenia množstva produktu na sklade

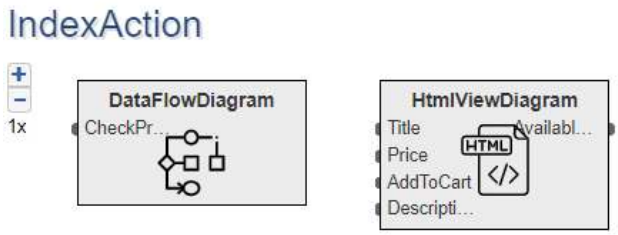
6.7. Prepojenie diagramov

Vyššie popísané diagramy boli navrhnuté s dôrazom na ich vzájomné prepojenie. Detaily realizácie týchto prepojení budú detailnejšie prezentované na metamodeloch v ďalšej kapitole venovanej vytvoreniu CASE nástroja implementujúceho navrhovanú metodológiu.

Vo WebGME sa k prepojeniam dostaneme prepnutím do vizualizéra *crosscut* na ľavej strane obrazovky. Avšak musíme sa nachádzať v príslušnom diagrame.

6.7.1. Prepojenie Controller diagramu s View a DataFlow diagramom

Ako už bolo spomenuté, *ControllerDiagram* môže obsahovať viacero *Controller*. Ten zas môže obsahovať akcie. Po prepnutí sa do tejto akcie jej vieme priradiť *DataFlow* diagram a taktiež *ViewDiagram*. Na Obr. 9 je znázornená index akcia kontroléra pre zobrazenie produktu. Ten obsahuje diagramy *DataFlow* a *View*.

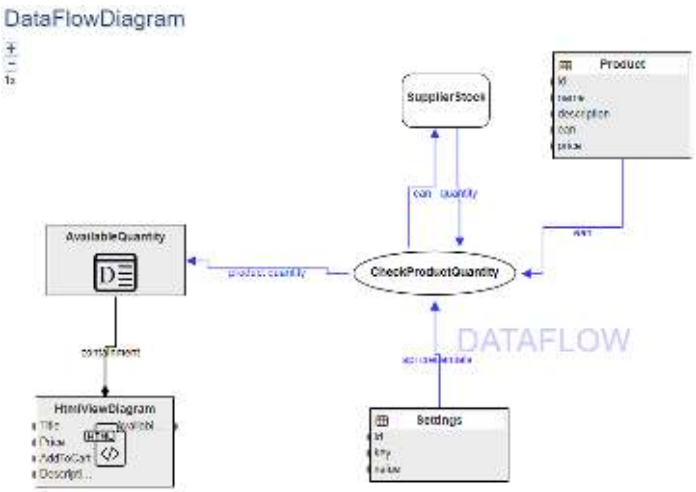


Obr. 9: Prepojenie *ControllerDiagram* s *DataFlowDiagram* a *ViewDiagramo*

6.7.2. Prepojenie DataFlow diagramu s Model a View diagramom

Toto prepojenie umožňuje sledovať spracovanie vstupných údajov z databázy, formulárov či externých entít a taktiež umožňuje sledovanie, kam tečú výstupy procesu. Môžu smerovať napríklad do databázy cez entitu *Model* diagramu alebo taktiež sa môžu dostať na stránku v podobe hodnoty v určitom elemente *View* diagramu. Pre otvorenie takéhoto pohľadu je potrebné sa v príslušnom *DataFlow* diagrame prepnúť do vizualizéra *crosscut.*

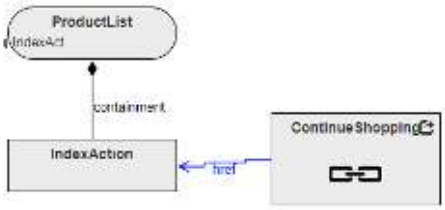
Zatiaľ čo na Obr. 8 bol *DataFlowDiagra*m bez prepojenia s inými diagramami značne redukovaný, tak na Obr. 10 je tento diagram kompletný. Obsahuje teda proces, ktorý načíta z databázy EAN produktu a prístupové údaje do API skladu dodávateľa. Následne s týmito údajmi zavolá API dodávateľa, ktorý vráti počet kusov na sklade. Tento počet je nakoniec priradený zodpovedajúcemu elementu šablóny GUI.



Obr. 10: Kompletný *DataFlowDiagram* zistenia množstva produktu na sklade

6.7.3. Prepojenie View diagramu s Controller diagramom

ViewDiagram môže obsahovať hypertextové prepojenia na určité akcie. Smer ich prepojení je možné taktiež sledovať vo vizualizéri *crosscut* v príslušnom *ViewDiagram*. Na Obr. 11 je zobrazený diagram hypertextových prepojení pre obrazovku nákupného košíka. Na ňom sa nachádza tlačidlo „Pokračovať v nákupe“, ktoré po kliknutí smeruje používateľa na stránku so zoznamom produktov. Tento model je obzvlášť užitočný, ak je v šablone viacero takýchto tlačidiel a prehľadne vizualizuje ich prepojenia.



Obr. 11: Hypertextové prepojenie *ViewDiagram* s *ControllerDiagram*

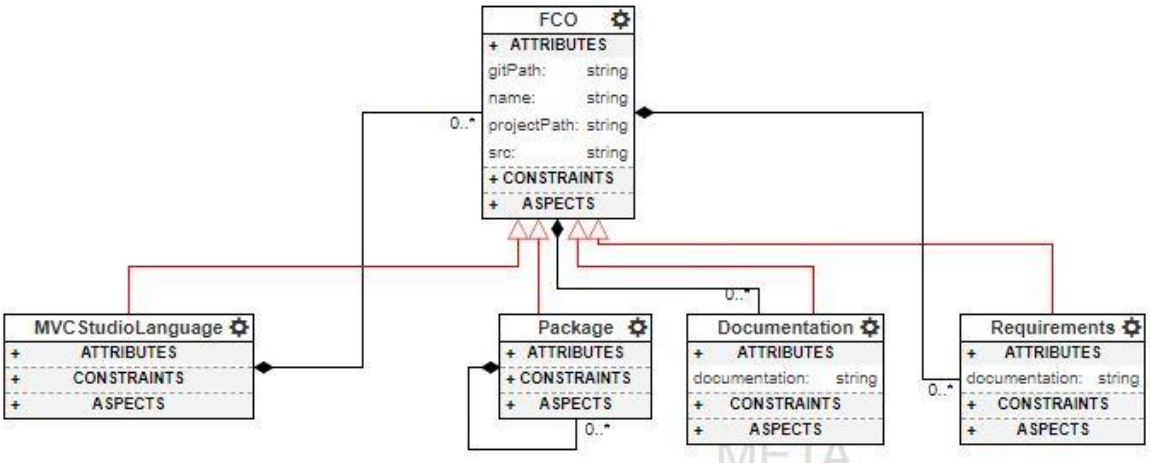
7. Implementácia metodológie a DSML do CASE nástroja MVCStudio pomocou metamodelovania

Obsahom tejto kapitoly je z vytvoreného DSML zostaviť príslušné metamodely, pomocou ktorých bude implementovaná metodológia modelovania webových systémov založených na architektúre MVC. Základom pre vytvorenie CASE nástroja MVCStudio bude metaCASE platforma WebGME.

Celkový metamodel systému bol pre zachovanie prehľadnosti rozdelený do viacero malých ucelených metamodelov. Keďže jednotlivé metamodely sú medzi sebou prepojené, tak objekty, ktoré patria určitému metamodelu sú farebne zvýraznené rovnakou farbou. To znamená, že keď v metamodeli je objekt inej farby, tak pochádza z iného metamodelu a je sem pridaný za účelom prepojenia týchto metamodelov.

7.1. Základný metamodel

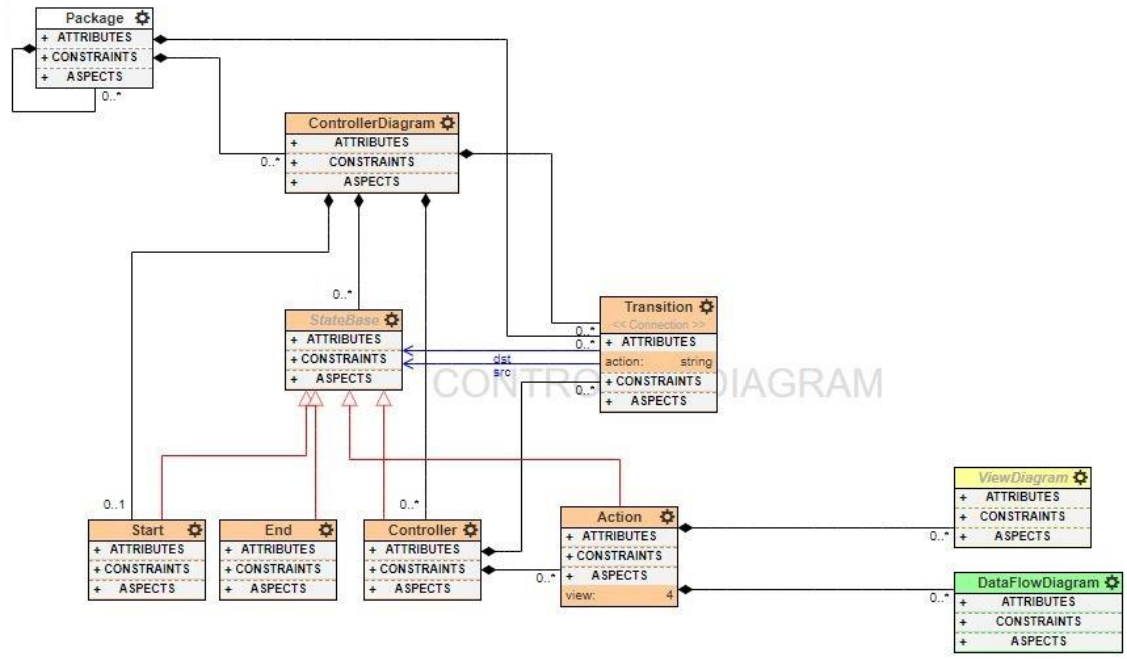
Tento metamodel je znázornený na Obr. 12. Obsahuje objekt *FCO – first class object,* ktorý tvorí koreň stromu dedičnosti. Ďalej je tu objekt *MVCStudioLanguage*, ktorý v kompozícii kvôli prehľadnosti združuje všetky objekty modelovacieho jazyka. Následne je tu objekt *Documentation*, ktorý umožňuje vytvárať dokumentáciu v ľubovoľnom kontexte pre ľubovoľný model, diagram alebo jeho uzol, čo zabezpečuje prepojenie s uzlom *FCO* s kardinalitou 0:N. Objekt *Requirements* umožňuje zaznamenávať textové požiadavky v ľubovoľnom kontexte rovnako ako *Documentation*. Posledným objektom je tu objekt *Package,* ktorý umožňuje lepšie štrukturovať projekt pomocou balíčkov, ktoré môžu byť vnorené, pričom balíček môže obsahovať ďalší balíček a tým tvoriť stromovú hierarchiu zlučujúcu navzájom súvisiace modely.



Obr. 12: Základný metamodel

7.2. Metamodel ControllerDiagram

Na Obr. 13 je zobrazený metamodel pre *ControllerDiagram,* pomocou ktorého sa modelujú rozličné stavy aplikácie. Ten obsahuje objekt *ControllerDiagram* reprezentujúci samotný diagram. Je prepojený s *Package* čo znamená, že balíček môže obsahovať viacero týchto diagramov. Ďalej sú s ním prepojené objekty, ktoré tento diagram obsahuje, ktorými sú *Controller, Action, Start, End.* Tieto objekty dedia z objektu *StateBase,* ktorý je prepojený s prepojovacím objektom *Transition*. Toto prepojenie teda zdedia všetky komponenty, čo umožňuje ich vzájomné prepájanie v modeli diagramu. Nachádza sa tu taktiež prepojenie na ďalšie diagramy. To umožňuje do objektu *Action* vytvoriť viaceré objekty *ViewDiagram* a *DataFlowDiagram.*

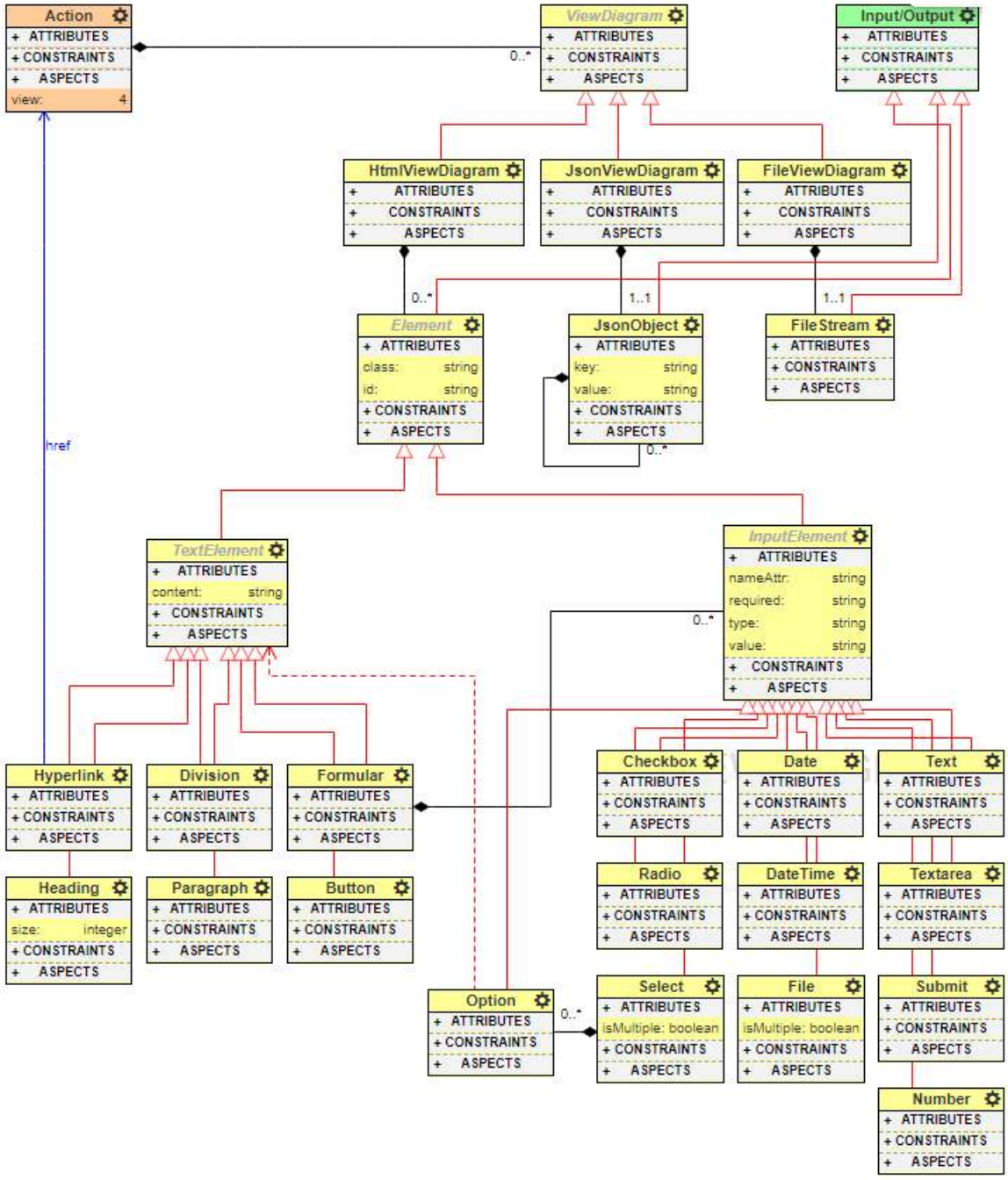


Obr. 13: Metamodel ControllerDiagram

7.3. Metamodel View diagram

Na Obr. 14 je znázornený metamodel *View* diagramu pre modelovanie pohľadu a teda aj používateľského rozhrania. Ten obsahuje abstraktný objekt *ViewDiagram* zlučujúci všetky komponenty diagramu. Od neho dedia všetky konkrétne druhy *View* diagramov, a to je *FileViewDiagram,* ktorý môže obsahovať *FileStream*, potom je tu *JsonViewDiagram,* ktorý obsahuje *JsonObject* a posledným najvyužívanejším je *HtmlViewDiagram,* ktorý obsahuje objekty typu *Element*. Tie sa ďalej špecifikujú v rámci hierarchie dedičnosti na konkrétne HTML komponenty. Čo sa týka prepojení, tak objekt typu *Hyperlink* má prepojenie s *Action,* čo umožňuje vytvárať hypertextové prepojenia medzi odkazom a akciou (object typu *Action* z diagramu

Controller). Taktiež je tu objekt typu *Input/Output* z metamodelu diagramu *DataFlow,* od ktorého dedia určité objekty. To umožňuje prepojiť *DataFlow* diagram s *View* diagramom.

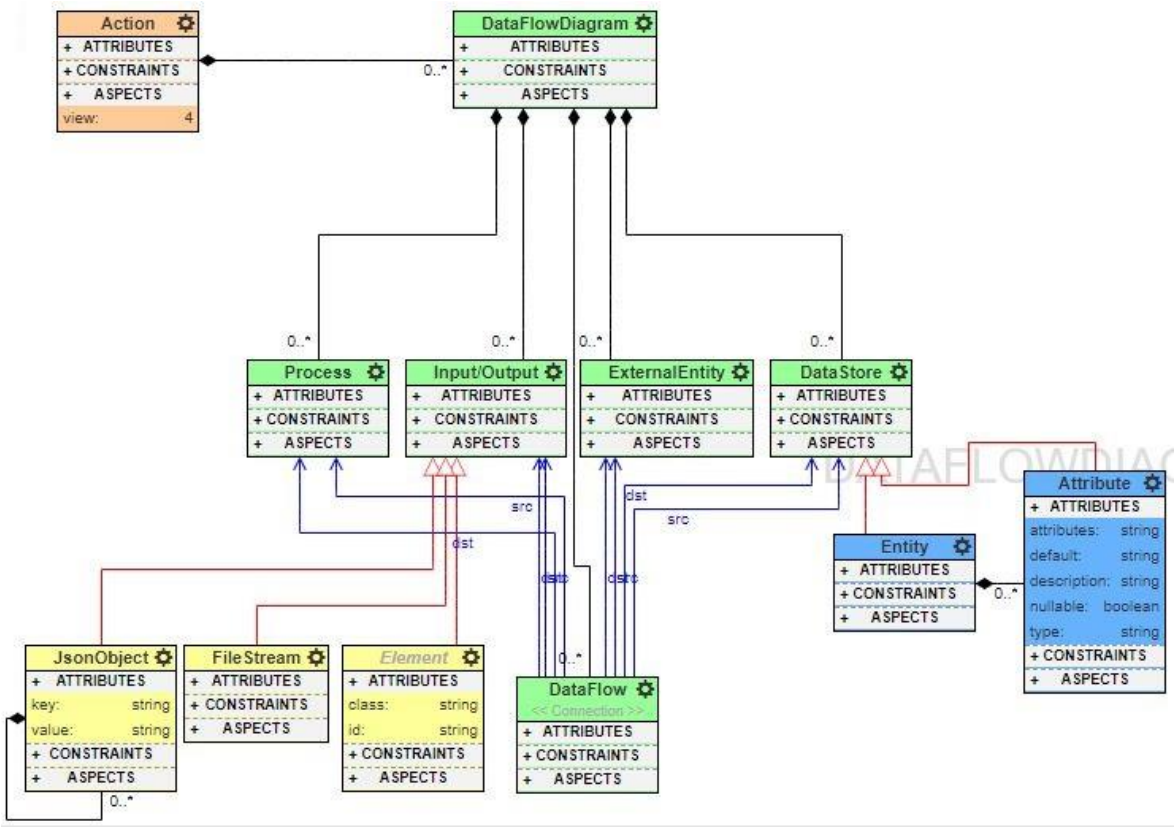


Obr. 14: Metamodel View diagram

7.4. Metamodel DataFlowDiagram

Tento metamodel zobrazený na Obr. 15 obsahuje *DataFlowDiagram* modelujúci dátový tok aplikácie. Obsahuje komponenty ako je *Proces, Input/Output, ExternalEntity, DataStore*. Tieto je možné v modeli prepájať vďaka objektu *DataFlow*, ktorý má s nimi prepojené ukazovatele. Čo sa týka prepojení, tak *DataFlowDiagram* je prepojený s *Action,* čo umožňuje prepojiť *DataFlow*

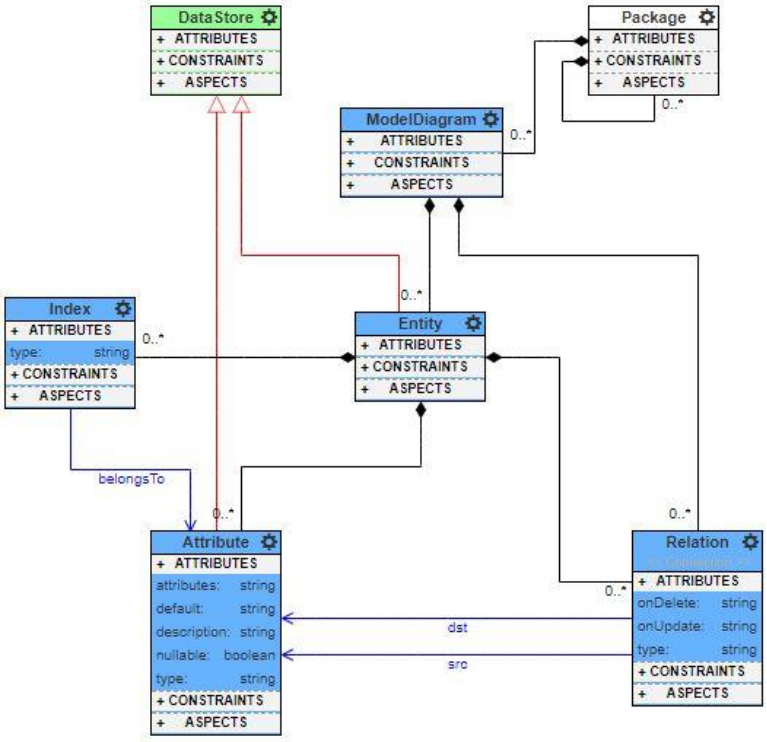
diagram s *Controller* diagramom. Ďalej od objektu *DataStore* dedia objekty *Entity a Attribute,* čo má za následok prepojenie *DataFlow* diagramu s *Model* diagramom. Nakoniec sú tu žlté objekty z *View* diagramu, ktoré dedia od objektu *Input/Output*, čo zabezpečuje prepojenie *DataFlow* diagramu s *View* diagramom.



Obr. 15: Metamodel DataFlow diagramu

7.5. Metamodel Model diagram

Ako posledný metamodel je na Obr. 16 znázornený *Model* diagram reprezentujúci dátovú vrstvu aplikácie. Ten obsahuje objekt *ModelDiagram*, ktorý je prepojený s objektom *Package*, čiže balíček projektu môže obsahovať tento diagram. Ďalej obsahuje objekty *Entity a Attribute*. Objekt *Index* je prepojený s *Attribute*, čo umožňuje definovať indexy pre určitý stĺpec databázy. Objekt *Attribute* je taktiež prepojený s objektom prepojenia *Relation,* čo umožňuje vytvárať prepojenia vo forme cudzích kľúčov. Nachádza sa tu aj objekt *DataStore* z *DataFlowDiagram* metamodelu. Od neho dedia *Entity a Attribute,* čo umožňuje v DataFlow diagrame prepájať procesy s databázovými entitami a taktiež aj jej atribútmi.



Obr. 16: Metamodel Model diagramu

8. Metodológia a návod na použitie CASE nástroja MVCStudio

Obsahom tejto kapitoly je stručne zhrnúť použitie vytvorenej metodológie a DSML implementovaného do CASE nástroja MVCStudio. V prílohe C sa nachádzajú ďalšie kompletné a podrobné informácie k tejto kapitole.

Pri návrhu a modelovaní systému sa potrebujeme zamerať na jeho kritické časti. V tomto prípade tieto časti sú dátová vrstva, riadenie stavov, GUI a dátový tok.

Neexistuje úplne univerzálny postup v akom poradí je potrebné takýto systém navrhovať a zostavovať jednotlivé diagramy, avšak odporúčame sa inšpirovať naším odporúčaným postupom a jednotlivé kroky prispôsobovať pre potreby konkrétnej situácie:

1. Na základe definícií požiadaviek na systém je potrebné sa zamyslieť nad dátovou vrstvou a namodelovať ju. Ak sa v prvotnom návrhu budú objavovať určité chyby a nezrovnalosti, nie je to problém, pretože kedykoľvek je možné tieto modely upraviť. Ak na začiatku ešte nie je jasná štruktúra dát, treba namodelovať aspoň to, čo je známe a ostatné elementy budú doplnené priebežne počas návrhu a zostavovania iných modelov.
2. Z definícií požiadaviek na systém je potrebné odhaliť hlavné funkcie systému. Tieto funkcie je potrebné rozdeliť do navzájom súvisiacich skupín. Zvyčajne sa rozdeľujú najprv na väčšie funkčné skupiny, ktoré sa nazývajú moduly (administrácia, profil používateľa, predný modul, atď.).
3. Následne pre každý modul znova skúmame funkcie systému a opäť sa ich snažíme rozdeliť do vzájomne súvisiacich menších skupín. Tieto skupiny môžeme chápať ako stavy, v ktorých sa používateľ nachádza (sekcia importu záznamov, výber šablóny, generovanie PDF, …). V rámci stavov môžu byť definované aj podstavy, ktoré označujeme ako akcie. Ich význam spočíva v podpore čiastkových funkcií poskytovaných daným kontrolérom (nahranie súboru).
4. Ďalším krokom je navrhnutie a namodelovanie GUI a iné typy pohľadov (HTML šablóna, súbor, JSON string). Tieto pohľady obsahujú pre daný typ špecifické komponenty. Z týchto komponentov vyskladáme požadovaný pohľad. Teda napríklad v prípade modelovaní GUI šablóny modelujeme formuláre, vstupné polia, odkazy, tlačidlá a veľa ďalších elementov.
5. V tomto momente máme k dispozícii a namodelované všetko čo potrebujeme pre zostavenie diagramu toku údajov. Je vhodné mať už zostavené predchádzajúce diagramy, pretože ich využíva vo veľkej miere. Preto odporúčame tento typ diagramu modelovať úplne na záver. Dátový tok dokáže prepájať dátovú vrstvu s pohľadovou vrstvou prostredníctvom procesov v kontroléroch.

9. Rozšírenie CASE nástroja MVCStudio

Vytvorený CASE nástroj zatiaľ umožňuje dizajnérovi manuálne modelovanie systémov. Táto kapitola je venovaná rozšíreniu navrhnutého CASE nástroja MVCStudio založeného na prostredí WebGME. Keďže rozširovanie prostredia WebGME si vyžaduje zmeny a doplnenie zdrojových kódov systému, je potrebné mať zavedenú jeho vlastnú inštaláciu. Postup je popísaný v systémovej príručke v prílohe B.

V nasledujúcom texte sú navrhnuté doplnky tohto nástroja za účelom vylepšenia, uľahčenia alebo automatizácie prác vývojárov na navrhovanom systéme.

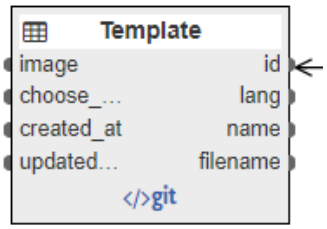
9.1. Implementované rozšírenia

Do vytvoreného CASE nástroja boli implementované nasledujúce rozšírenia.

9.1.1. Vykonateľné modely - prepojenie modelov so softvérom

Jednotlivé artefakty v modeloch CASE systému môžu byť v cieľovom systéme reprezentované konkrétnymi zdrojovými kódmi. Účelom tohto rozšírenia je rýchlo a jednoducho sa dostať od artefaktu k jeho implementácii v podobe zdrojového kódu cieľového systému. Toto rozšírenie je prínosom hlavne pre vývojárov daného systému tým, že im šetrí čas pri hľadaní implementácie tohto modelu v zdrojových kódoch, ktoré zvyčajne môžu byť veľmi rozsiahle.

Táto funkcia bola implementovaná do artefaktov navrhovaného CASE nástroja v podobe tlačidiel v jeho spodnej časti uzla modelu viď. Obr. 17.



Obr. 17: Tlačidlá pre prechod ku zdrojovým kódom artefaktu

Zdrojové kódy je možné otvoriť dvoma spôsobmi:

- **git** - po kliknutí na tlačidlo s piktogramom ***git*** sa vo webovom prehliadači otvorí nová karta zobrazujúca zdrojový kód aktuálneho artefaktu vo webovom rozhraní systému pre správu verzií *gitlab*. Táto možnosť je vhodná, ak vývojár nemá vo svojom aktuálnom zariadení zavedené zdrojové kódy cieľového systému alebo len kvôli rýchlemu nahliadnutiu nechce spúšťať celé vývojové integrované prostredie IDE;

- **vývojové prostredie** – po kliknutí na piktogram **</>** sa otvorí zdrojový kód artefaktu vo vývojovom prostredí. Táto možnosť je dostupná pre zariadenia s operačným systémom typu Microsoft Windows.

Pre realizáciu tejto funkcie bolo potrebné vytvoriť vlastný protokol *editor://*, ktorý prenáša identifikačné údaje zdrojového kódu z webového prehliadača do hosťujúceho operačného systému.

Tento protokol vyzerá nasledovne:
editor://open/?file=APP_ROOT\app/Template.php&line=1

Po stlačení tlačidla sa tento protokol vyvolá a hosťujúci operačný systém ho odchytí a posunie preddefinovanému programu na ďalšie spracovanie. Tento program na základe používateľskej konfigurácie vyberie preferované vývojové prostredie a spustí ho spolu s príkazom na otvorenie zdrojového kódu na základe identifikátorov obsiahnutých v prijatom protokole. Riešenie je pripravené pre väčšinu moderných v súčasnosti využívaných vývojových prostredí a editorov zdrojových kódov používaných pre vývoj webových aplikácií, pričom je jednoduché nakonfigurovať podporu aj na ďalšie vývojové prostredia. Aktuálne podporované editory a vývojové prostredia sú:

- PhpStorm;
- NetBeans;
- Nusphere PHPEd;
- SciTE;
- EmEditor;
- PSPad Editor;
- gVim;
- Sublime Text 3.

Pred prvým použitím je potrebné nainštalovať a nakonfigurovať obslužný program hostiteľského počítača. Návod s postupom pri tejto inštalácii je umiestnený v systémovej príručke v prílohe B.

Tento nástroj dokáže otvoriť zdrojový kód s kurzorom na definovanom riadku, avšak táto funkcia nie je v tomto nástroji plne využitá (prednastavený je prvý riadok), no v budúcnosti je možné prostredníctvom tejto funkcie nasmerovať programátora na určitú metódu alebo dokonca určitú problematickú konštrukciu zdrojového kódu, ktorá môže byť predmetom jeho záujmu.

9.1.2. Generovanie modelov z textovej analýzy požiadaviek

Analytickú fázu vykonáva programátor manuálne, pričom si v hlave spracováva zoznam požiadaviek a vytvára predstavu, aké artefakty budú v určitom modeli, aké budú mať vlastnosti alebo akým spôsobom budú prepojené s ostatnými uzlami modelu.

Pre zefektívnenie tejto fázy bol implementovaný automatizačný krok pomocou rozšírenia **ArtifactGenerator**, ktorý zo zoznamu požiadaviek dokáže automaticky vygenerovať predbežný návrh nových chýbajúcich systémových objektov. Vychádzame z toho, že zoznamy požiadaviek sú zozbierané vo forme textových požiadaviek. Tieto požiadavky sú vety v určitom jazyku. Táto veta obsahuje podstatné mená, slovesá a prídavné mená, ktoré je možné extrahovať z textu na základe slovno-druhovej analýzy. Z extrahovaných podstatných mien je možné vytvoriť napríklad v objektovo orientovanom programovaní objekty, triedy alebo aktérov. Slovesá hovoria o funkcionalite, teda môže ísť napríklad o metódy, procesy alebo činnosti, ktoré znovu smerujú k ďalším artefaktom. Prídavné mená môžu indikovať vlastnosti a stavy konkrétnych artefaktov.

Syntéza modelov softvérového systému si vyžaduje vysoký stupeň intelektu s detailnými vedomosťami o cieľovej implementačnej doméne a teda je v tomto bode potrebný zásah človeka. V ďalšom kroku dizajnér rozhoduje, ktoré z vygenerovaných artefaktov sú relevantné pre použitie a pracuje s nimi ďalej, pričom nevyhovujúce artefakty môže odstrániť.

Princíp fungovania je taký, že do ľubovoľného modelu je potrebné pridať objekt typu *Requirement*, ktorého účelom je uchovávanie textových požiadaviek pre diagram / objekt / model v aktuálnom kontexte. Pre vygenerovanie artefaktov z textových požiadaviek je potrebné spustiť **ArtifactGenerator** podľa Obr. 18. Nové analyzované artefakty budú automaticky pridané do modelu v aktuálnom kontexte.

Generátor artefaktov je možné použiť vo všetkých typoch modelov nástroja. Nasledujúca Tab. 2 zobrazuje, aké artefakty v ktorom modeli budú vytvorené. Teda napríklad v *DataFlowDiagrame* budú z podstatných mien vytvorené artefakty meta typu *DataStore* a zo slovesa bude vytvorený artefakt s meta typom *Process*. Tieto meta typy sú vysvetlené v kapitole 7.

Model	Podstatné meno	Sloveso
Package	ControllerDiagram	-
ModelDiagram	Entity	-
ControllerDiagram	Controller	-
Controller	-	Action
DataFlowDiagram	DataStore	Process
HtmlViewDiagram	Division	-
Formular	Text	-
JsonViewDiagram	JsonObject	-
JsonObject	JsonObject	-
Entity	Attribute	-

Tab. 2: Mapovanie generovania artefaktov zo slovných druhov pre modely

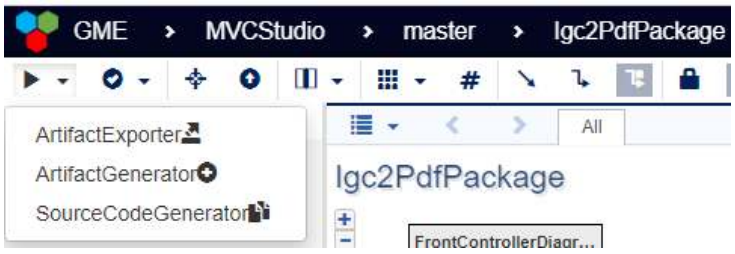
Prínos takejto automatizácie je v zrýchlení procesu modelovania systému a je tiež badateľný aj vo fáze verifikácie pomocou modelov. Pomenovania jednotlivých artefaktov (napr. komponenty GUI v diagrame obrazovky) budú prevzaté z textových požiadaviek, čím zákazník, ktorý bude verifikovať vytvorený návrh systému, bude rozumieť tomu čo jednotlivé artefakty reprezentujú.

9.1.3. Generovanie zdrojových kódov z modelov

Ďalším zo spôsobov prepojenia modelu so softvérom, je umožniť generovanie zdrojových kódov priamo na základe modelov nadizajnovaných v CASE systéme.

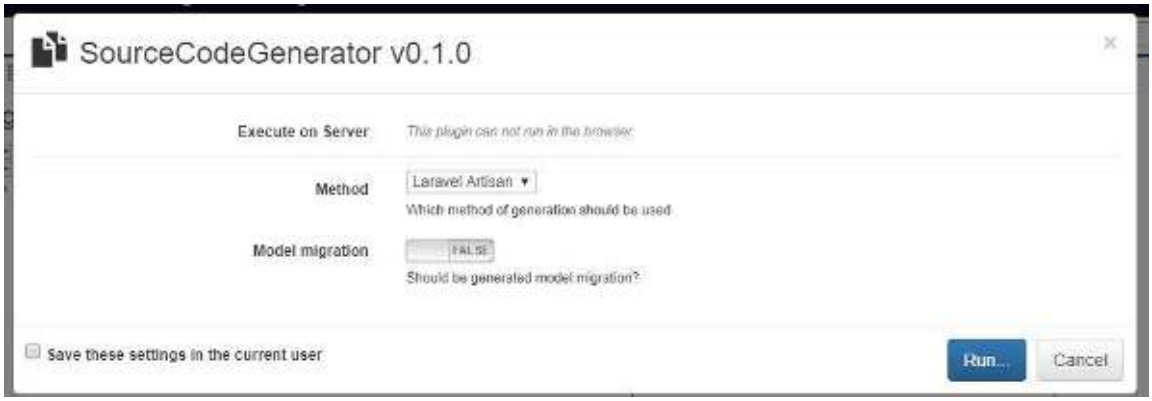
Pre tento účel boli implementované 2 rozšírenia prostredia WebGME:

- **ArtifactExporter –** jeho účelom je analyzovať všetky prvky nadizajnovaného modelu systému a vyexportovať dôležité informácie nielen pre potreby generovania zdrojových kódov vo forme JSON reťazca. Na Obr. 18 sa nachádza ponuka aktívnych rozšírení v aktuálnom kontexte, z ktorej je možné spustiť toto rozšírenie. Po úspešnom vykonaní sa v správe o výsledku nachádza vygenerovaný JSON súbor. Toto rozšírenie môže byť inicializované aj iným rozšírením, pričom preberie jeho výstup a ďalej s ním pracuje.



Obr. 18: Ponuka a spustenie aktívnych rozšírení

- **SourceCodeGenerator –** je zodpovedný za samotné generovanie zdrojových kódov. Pred jeho spustením v kontextovom okne viď. Obr. 19 je možné nastaviť metódu generovania zdrojových kódov a zapnúť generovanie databázových migračných súborov pre modely (M – model z MVC). Po spustení sa automaticky inicializuje rozšírenie *ArtifactExporter*, od ktorého si preberie schému nadizajnovaných artefaktov a vygeneruje z nej *kontroléry* a *modely,* prípadne aj databázové migračné súbory, ak sú vyžadované. Zdrojové kódy budú umiestnené na základe definovaných menných priestorov v rámci konkrétneho modelu do vývojovej verzie projektu softvérového systému, ktorá je definovaná v ROOT koreňovom uzli modelu projektu.



Obr. 19: Konfigurácia rozšírenia SourceCodeGenerator

Systémy alebo ich komponenty môžu byť vyvíjané v rôznych programovacích jazykoch alebo systémových rámcoch. Oddelenie modulu analýzy artefaktov modelu (opätovná použiteľnosť) od generovania kódov ma tú výhodu, že umožňuje vytvoriť špecifický generátor pre ľubovoľný programovací jazyk či systémový rámec bez nutnosti implementovania vlastnej modelovej analýzy. Teda vytvorenie nového generátora kódov bude výrazne rýchlejšie a jednoduchšie a nevyžaduje pokročilú znalosť jadra systému WebGME.

Pre zlepšenie dokumentácie softvéru môže generátor do týchto zdrojových kódov vkladať dôležité informácie z modelov vo forme komentárov a anotácií. Týmto je možné dosiahnuť vzájomné prepojenie všetkých projektových artefaktov od cieľov a požiadaviek až po implementácie, súvislosti a závislosti. Napríklad artefakt v modeli riešiaci určitú požiadavku na funkcionalitu systému by túto informáciu premietol priamo do zdrojového kódu vo forme anotácie. Taktiež by bola zlepšená navigovateľnosť medzi projektovými artefaktmi.

Obojsmerné prepojenie CASE nástroja a implementácie je možné dosiahnuť napríklad tým, že by CASE nástroj obsahoval rozšírenie na analýzu zdrojových kódov, ktoré by hľadalo dôležité poznatky v komentároch a anotáciách vložených programátorom. Ten by dôležité poznatky

vhodným spôsobom označil a CASE nástroj by tieto poznatky automatizovane natiahol do modelu a vhodným spôsobom ich vizualizoval.

9.2. Návrhy na ďalšie rozšírenia pre budúce vylepšenie

Užitočných rozšírení do vytvoreného CASE nástroja je možné navrhnúť veľmi veľa, no nezmestili by sa do rozsahu tejto diplomovej práce. Považujeme za vhodné niektoré návrhy aspoň načrtnúť a teda v ďalšom texte sú popísané ako inšpirácia pre budúce práce.

9.2.1. Hľadanie a zobrazovanie nekonzistencií

Ďalšou možnosťou ako prepojiť model s implementáciou je rozšírenie systému WebGME o modul, ktorý bude spracovávať chybové a prístupové logy generované webovými servermi, akými sú napríklad Apache alebo Nginx. Spracované logy budú vhodným spôsobom interpretované alebo vizualizované v určitých modeloch, ku ktorým sa vzťahujú. Logy sú vygenerované s určitou pevne definovanou štruktúrou, z ktorej je možné veľmi jednoduchým spôsobom parsovať jednotlivé informácie použitím napríklad regulárnych výrazov.

Ak sa v chybovom logu zobrazí problém, ktorý vznikol v určitom module, tak v CASE nástroji tento model bude indikovať neštandardnú situáciu s popisom problému a odkazom na všetky ďalšie súvisiace artefakty, či už iné modely a ich prvky alebo priamo zdrojové kódy. Programátor teda bude ihneď upozornený na výskyt neštandardnej situácie a bude ju môcť ihneď riešiť vďaka kontextu získanému z chybového logu vizualizovanom v modeli

Prístupový log by bolo možné využiť na určenie najviac a najmenej využívaných uzlov systému. V praxi by sa takáto analýza dala využiť pre naplánovanie škálovania najviac zaťažovaných komponentov a naopak degradáciu výpočtového výkonu pre menej namáhané komponenty. Taktiež to môže prispieť k identifikácii kritických častí systému z hľadiska používateľského používania. Prístupový log je častokrát aj veľmi dobrým zdrojom pre procesy odstraňovania chýb v softvéri nasadenom v produkčnom prostredí.

9.2.2. Odhaľovanie súvislostí

V praxi veľmi často prichádzajú určité zmenové, opravné alebo doplňujúce požiadavky na softvérový systém. Pre realizáciu týchto požiadaviek programátor využíva svoje znalosti o systéme alebo si pomáha dokumentáciou pričom ju musí manuálne prechádzať a hľadať podstatné informácie. Bolo by možné implementovať modul, ktorý analyzuje súvislosti v nadizajnovanom systéme a na vyžiadanie ich vizualizuje vývojárovi.

Niekedy je obrázok výhodnejší ako vysoký počet strán textu.

9.2.3. Prepojenie modelov so softvérom

Úspešný systém je taký, ktorý je možné prispôsobovať meniacim sa požiadavkám a podmienkam. Aby bolo systém možné meniť aj neskôr po jeho implementácii, je potrebné mať o ňom veľmi dobré znalosti. Tieto znalosti sa dajú čerpať aj z dokumentácie v forme modelov. Často sa však stáva, že modely po určitom čase prestávajú byť s implementáciou aktuálne a rozchádzajú sa. Dopomôcť tomu, aby model žil spolu s projektom môžeme tým, že vhodným spôsobom prepojíme tieto 2 strany využitím spätného inžinierstva RTE (Round-Trip Engineering).

9.2.4. Testovanie softvéru

Veľmi dobrou praxou je popri vývoji aplikácie si vytvárať testovacie sady, ktoré dokážu otestovať, či rôzne podmienky na vstupe vracajú správne výstupné hodnoty.

Do modelu by bolo implementované rozšírenie, ktoré dokáže automatizovane spúšťať tieto testy a ich výsledky preniesť do modelov a vhodným spôsobom ich vizualizovať. Rozšírenie sa môže prihlásiť na odber zmien informácii o zmenách v modeloch systému WebGME. Následne môže vývojára upozorniť o zmenených častiach systému, ktoré je potrebné znova otestovať, prípadne tieto testy spustiť automaticky a notifikovať vývojára v prípade neúspešných testov. Taktiež môže notifikovať vývojára o slabom pokrytí zdrojových kódov testmi.

10. Demonštrácia MVCStudio na príkladoch aplikácie igc2pdf

V tejto kapitole bude demonštrovaný vytvorený CASE nástroj MVCStudio implementujúci vytvorený DSML s využitím metodológie pre podporu vývoja webových aplikácií založených na MVC architektúre.

Ako bolo v práci identifikované, tak medzi kritické časti systémov tejto kategórie patrí dátová vrstva, riadenie stavov, GUI a dátový tok. Pri demonštrácii budeme postupovať v súlade s metodickým postupom navrhnutým v kapitole 8.

V hlavnej časti diplomovej práce si ukážeme po jednom príklade pre každú kritickú časť aplikácie. V prílohe D sa nachádza kompletná detailná demonštrácia systému spolu s jeho špecifikáciou a definovaním jeho požiadaviek.

10.1. Popis aplikácie igc2pdf

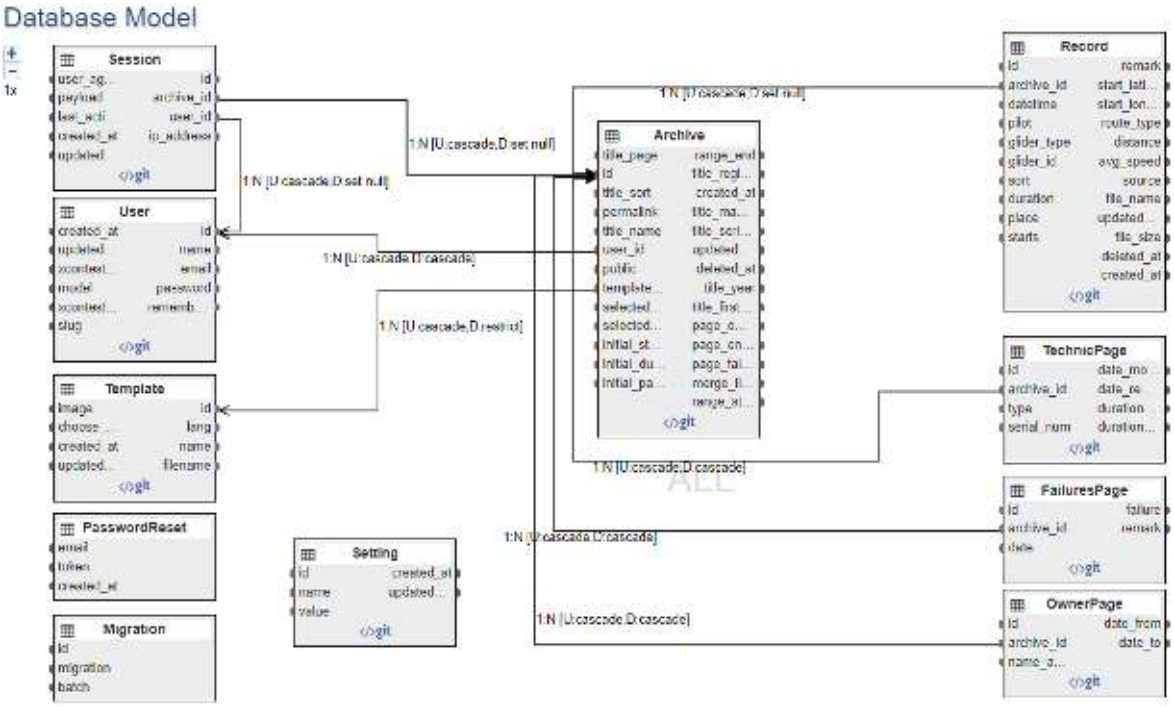
Je to webová aplikácia, ktorej účelom je zjednodušenie administratívnych úkonov pilotom LŠZ pri vedení letových dokumentácií. Tento proces je legislatívne vyžadovaný LAA SR. Aplikácia umožňuje pilotovi importovať svoje letové záznamy v .IGC súboroch alebo cez import z leteckého portálu xcontest.org. Tieto záznamy je ďalej možné modifikovať a prispôsobovať rôznymi spôsobmi. Používateľ si na základe zvolenej šablóny stiahne požadovanú letovú dokumentáciu vo formáte PDF. Využiť môže taktiež viaceré možnosti archivácie, čo umožňuje kontinuálne vedenie letovej dokumentácie pri budúcich letoch.

Táto aplikácia bola navrhnutá a zrealizovaná v roku 2018 v rámci bakalárskej práce [1]. Video ukážka aplikácie sa nachádza na odkaze [26]. V popise videa sa nachádza aj aktuálny odkaz na aplikáciu nasadenú v produkčnom prostredí. Odporúčame prehrať si toto video (3min) pred pokračovaním.

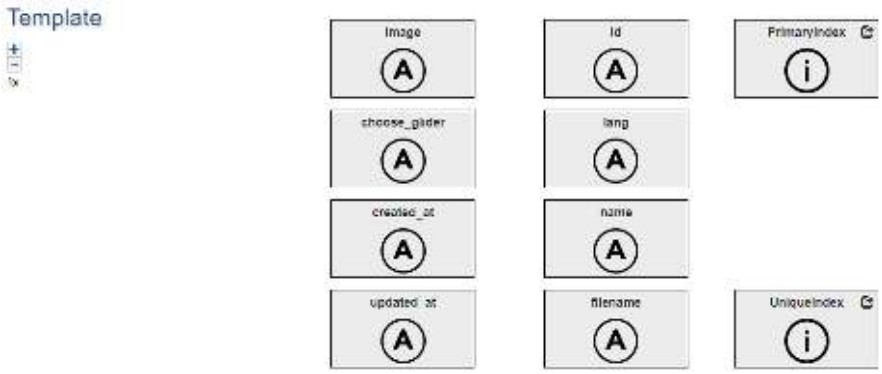
10.2. Dátová vrstva

Na Obr. 20 sa nachádza *ModelDiagram* zachytávajúci dátovú vrstvu, ktorú systém využíva pre účely manipulovania a perzistovania informácií. Obsahuje jednotlivé databázové entity s ich atribútmi a zároveň zobrazuje prepojenia medzi nimi spolu s definovaním kardinality prepojenia a pravidiel zachovania RI pre akcie aktualizácie a vymazania kľúčového záznamu.

Každá entita má taktiež namodelované aj svoje atribúty a entity. Ako príklad si uvedieme tabuľku *Template,* ktorej model je zachytený na Obr. 21. Táto tabuľka má definované okrem atribútov aj index primárneho kľúča pre atribút *id* a taktiež index pre zabezpečenie unikátnej hodnoty pre atribút *filename.*



Obr. 20: *ModelDiagram* databázovej vrstvy



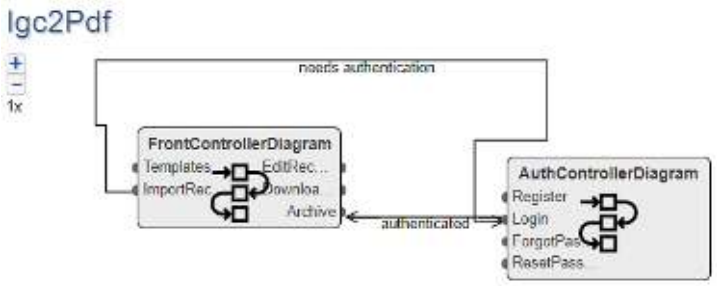
Obr. 21: Atribúty a indexy entity *Template*

10.3. Modelovanie systémových modulov

Aplikácia bola pri návrhu rozdelená na 2 hlavné celky, ktoré sú reprezentované systémovými modulmi.

Ako je možné vidieť na Obr. 22, tak jedným s týchto modulov tvorí *Front*, ktorý zapuzdruje hlavnú biznis logiku aplikácie dostupnú každému návštevníkovi webovej stránky.

Druhým modulom je *Auth*, ktorý zapuzdruje celú logiku okolo autentifikácie. Konkrétne ide o mechanizmus prihlásenia a registrácie. V prípade, že používateľ zabudne svoje heslo, má k dispozícii aj možnosť vyžiadať si zmenu hesla a po overení jeho identity mu je umožnené nastavenie nového hesla.

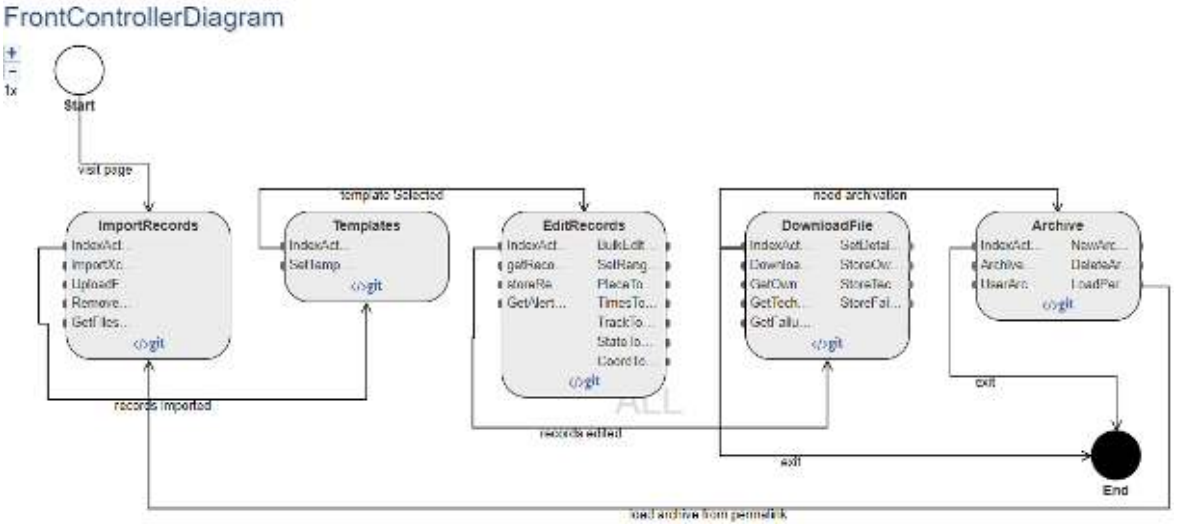


Obr. 22: Modelovanie systémových modulov

10.4. Modelovanie stavov hlavnej časti systému

Hlavnú časť systému reprezentujú všetky funkcie a obrazovky prístupné každému používateľovi bez obmedzenia. Jeho model sa nachádza na Obr. 23. Sú tu namodelované hlavné funkčné celky zlúčené do jednotlivých kontrolérov. Prechod medzi jednotlivými kontrolérmi je namodelovaný prepojením s popisom, kedy dochádza ku prechodu a teda ku zmene stavu aplikácie. Je tu taktiež definovaný počiatočný a koncový stav. V prílohe D sa nachádzajú aj fotky obrazoviek týchto stavov.

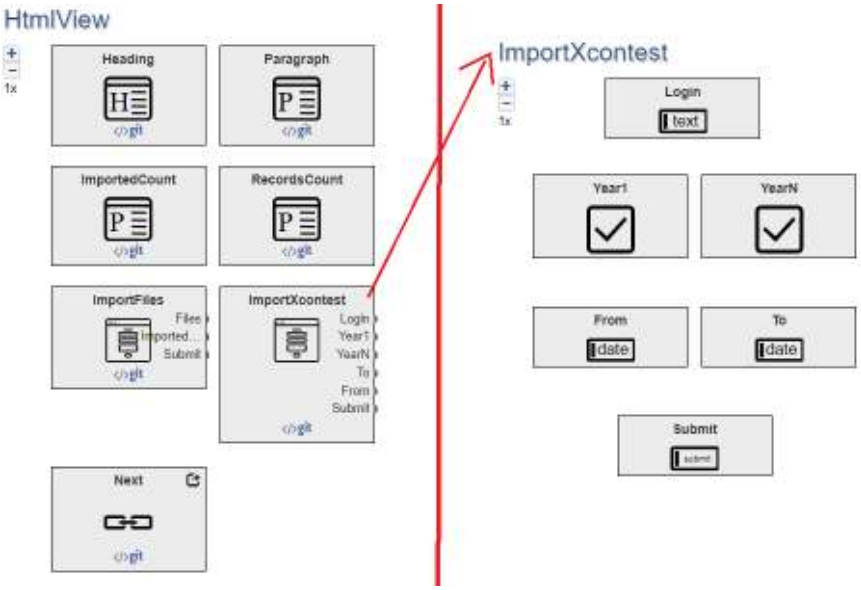
Po návšteve stránky sa používateľ dostane na obrazovku, kde môže importovať svoje letové záznamy. Potom nasleduje obrazovka s výberom šablóny záznamníka. Po výbere sa dostane v ďalšom kroku do sekcie pre úpravu letových záznamov. Ďalším krokom je obrazovka s možnosťou nastavenia a stiahnutia dokumentácie v PDF. Z tejto obrazovky môže prácu buď ukončiť alebo pokračovať k funkciám archivácie na ďalšej obrazovke. Po ukončení archivácie prácu môže ukončiť. Tento kontrolér sa taktiež stará o načítanie archívu z trvalého odkazu, pričom používateľ je presmerovaný na úvodnú obrazovku.



Obr. 23: FrontControllerDiagram – stavový model hlavných funkcií systému

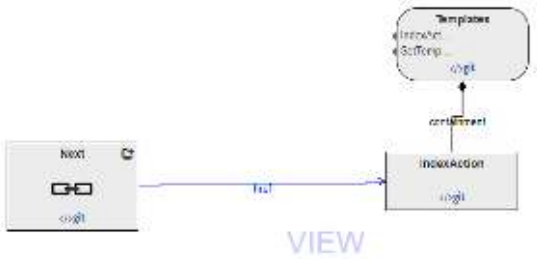
10.5. Modelovanie pohľadu

Pre ukážku pohľadu bola zvolená úvodná obrazovka kontroléra *ImportRecords.* Šablóna tejto obrazovky je namodelovaná na Obr. 24. Táto obrazovka teda obsahuje nejaký nadpis, podnadpis, počítadlo záznamov, tlačidlo pre pokračovanie na ďalší krok, formulár pre nahrávanie súborov a formulár pre import z portálu xcontest.org. Model tohto formulára nachádzajúci sa na pravej strane obrázka obsahuje ďalšie HTML elementy. Takto sú modelované všetky formuláre.



Obr. 24: ViewDiagram šablóny pre vykreslenie akcie index kontroléra *ImportRecords*

Po prepnutí do vizualizéra *crosscut* viď. Obr. 25 sa objaví model vizualizujúci smer prepojení hypertextových odkazov. Takže tlačidlo pokračovať v tomto prípade smeruje na ďalší krok do kontroléra *Templates*.

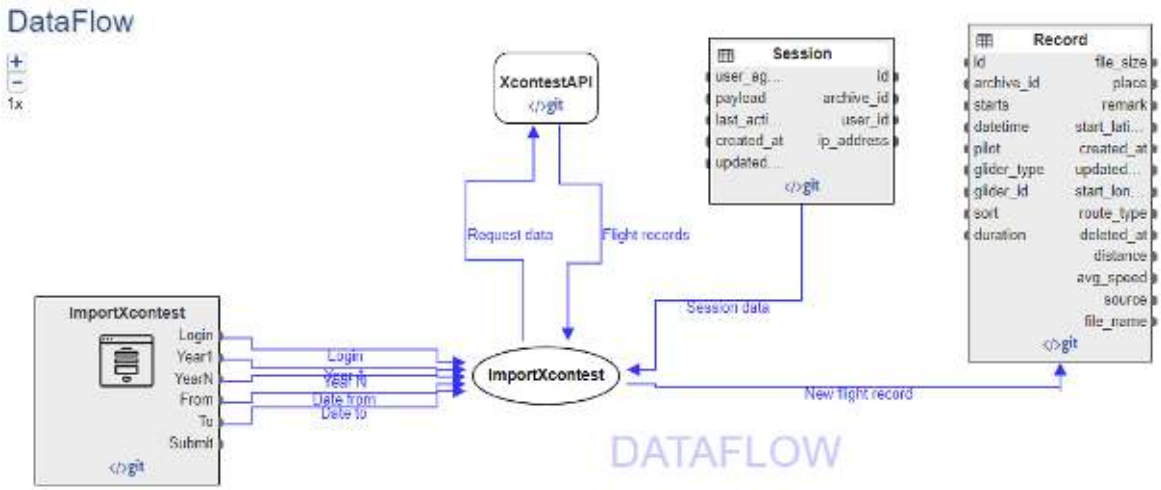


Obr. 25: Model hypertextového prepojenia v šablóne kontroléra *ImportRecords*

10.6. Modelovanie dátového toku

Pre ukážku modelovania toku dát viď. Obr. 26 sme si zvolili funkciu pre import letových záznamov z portálu xcontest.org, ktorá je súčasťou kontroléra *ImportRecords*.

Z modelu môžeme vidieť, že proces preberie potrebné identifikácie pilota z formulára (artefakt z diagram *View*) a databázovej entity *Session* (diagram *Model*). Z týchto dát je vyskladaná požiadavka, ktorá je odoslaná na externú entitu API servera. Server vráti zoznam letových záznamov, ktorý je spracovaný vhodným spôsobom a perzistovaný do databázovej entity *Record.*



Obr. 26: Model toku dát importu letových záznamov z portálu xcontest.org

10.7. Ukážka generovania artefaktov z textových požiadaviek

Pre potreby ukážky generovania modelov z textových požiadaviek sme vybrali databázovú entitu *User* reprezentujúcu zaregistrovaného používateľa aplikácie.

Ako je možné vidieť na Obr. 27, objekt *Requirements* obsahuje zmenovú požiadavku v textovej podobe. Konkrétne ide o požiadavku, aby pri používateľovi bolo možné uchovávať jeho identifikátory externej služby xcontest. Po spustení pluginu *ArtifactGenerator* sú požiadavky analyzované a na základe určitých pravidiel sú vytvorené nové objekty tohto modelu. V tomto prípade tieto uzly sú umiestnené pod požiadavkami. Ide o *xcontest_id, model, xcontest_login* a *slug.* Vývojár teraz rozhodne, čo je s týmito objektmi potrebné ďalej robiť. Objekt *model* v tomto kontexte nedáva žiaden zmysel, takže ho vývojár odstráni.

Obr. 27: Generovanie artefaktov entity *User.*

Záver

V tejto diplomovej práci sme sa na začiatku venovali analýze verifikácie a validácie požiadaviek na softvérové systémy, pričom sme sa taktiež zamerali na aktuálny stav v tejto oblasti. Dospeli sme k záveru, že procesy verifikácie a validácie sú kriticky dôležité pre zaistenie, aby vyvíjaný softvér spĺňal dôležité vopred dohodnuté parametre. Kľúčové je, aby sa tieto procesy začali robiť tak skoro, ako je to len možné a taktiež kontinuálne počas celého SwLC.

V ďalšej časti práce sme pokračovali analýzou modelovania, pričom sme uviedli, že zvyšovanie úrovne abstrakcie môže viesť ku rapídnemu nárastu produktivity vývoja softvéru. Analyzovali sme jazyk UML, ktorý sa stal štandardom pre systémové modelovanie, pričom sme popri jeho výhodách uviedli aj z nášho pohľadu najväčšiu nevýhodu. Tá spočíva v snahe byť univerzálnym nástrojom pre všetkých a pre všetko, čo neumožňuje ďalej dvíhať úroveň abstrakcie. Taktiež univerzálny nástroj nemusí podporovať riešenie špecifických problémov optimálnym spôsobom. Analyzovali sme taktiež DSM a došli sme k záveru, že pre riešenie špecifických systémov môže byť vhodnou alternatívou k UML najmä vďaka tomu, že zvyšovanie abstrakcie je jeho základným prvkom. Taktiež je navrhnutý s cieľom riešiť špecifické problémy čo najefektívnejšie.

Následne po analyzovaní problematiky metamodelovania a jeho výhod sme dospeli k záveru, že metamodelovanie je možné využiť pre potreby DSM. Metamodelovanie umožňuje zadefinovať DSML pre modelovanie špecifickej domény s požadovanou úrovňou abstrakcie a detailov.

Ďalej sme analyzovali platformu generického modelovania WebGME, pričom sme zhodnotili, že tento nástroj umožňuje vyvinutie CASE nástroja implementujúceho metodiku DSM. Vhodným nakonfigurovaním jeho metamodelov sa zadefinuje vlastný DSML modelovací jazyk. Taktiež tento nástroj dokáže priniesť vlastné CASE riešenia pre stredné a malé firmy, pretože vytvorenie vlastného nástroja na mieru je pre nich finančne a časovo nerentabilné.

Ďalším krokom práce bolo vytvorenie DSML a metodológie pre návrh a modelovanie systémov špeciálnej kategórie. Do tejto kategórie patria webové aplikácie založené na MVC architektúre, ktorá je určená na vývoj interaktívnych systémov a je v oblasti webových systémov veľmi rozšírená. Pri návrhu metodológie sme vychádzali z toho, že metodológia musí podporovať modelovanie všetkých kritických častí takejto aplikácie. V analýze sme odhalili, že medzi tieto kritické časti patri dátová vrstva, GUI, riadenie stavov a riadenie dátového toku. Vytvorené modely novej metodológie boli pre lepšie pochopenie ilustrované na príklade zjednodušeného internetového obchodu.

Následne bol vytvorený modelovací jazyk DSML spolu s jeho metodológiou implementovaný do platformy WebGME, čím bol vytvorený nový CASE nástroj s názvom MVCStudio.

Pri pokuse o vylepšenie efektivity vytvoreného CASE nástroja boli navrhnuté a implementované viaceré rozšírenia. Patrí medzi nich generovanie zdrojových kódov kostry kritických častí systému, generovanie prvotného návrhu modelov z textových požiadaviek a prechod jedným klikom od modelu k jeho reprezentácii v zdrojovom kóde v gitlabe, alebo vývojovom prostredí IDE. Taktiež boli navrhnuté aj ďalšie užitočné rozšírenia pre budúce práce.

Bola vypracovaná metodika, ktorá definuje odporúčaný postup, ako postupovať pri použití vytvorenej metodológie a vytvoreného CASE nástroja MVCStudio.

Na záver bol CASE nástroj MVCStudio spolu s metodikou na jeho použitie demonštrovaný na webovej aplikácii igc2pdf, ktorej účelom je, aby uľahčila pilotom LŠZ procesy elektronickej evidencie letových záznamov a generovanie legislatívou vyžadovaných záznamníkov vo formáte PDF. Táto aplikácia bola vytvorená v rámci bakalárskej práce.

Zoznam použitej literatúry

[1]. FERENČÍK FRANTIŠEK. Spracovanie letových záznamov v IGC formáte do textových dokumentov. Bakalárska práca. [Online] Technická univerzita v Košiciach, 25.05.2018. [cit. 2019-12-03]. Dostupné na internete: https://opac.crzp.sk/?fn=detailBiblioForm&sid=5301BA0C1F12FE31B5404AA915FD.

[2]. PROFESSOR F. L. BAUER. Nato software engineering conference 1968, Garmisch Nemecko, [cit. 2019-03-15], Dostupné na internete: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.

[3]. SOMMERVILLE IAN. Software engineering / Ian Sommerville. — 9th ed., 2011, ISBN-10: 0-13-703515-2, Dostupné na internete: http://dinus.ac.id/repository/docs/ajar/RPL-Sommerville_-_Software_Engineering_9ed.pdf.

[4]. STN ISO/IEC 12207: Systémové a softvérové inžinierstvo. Procesy životného cyklu softvéru. Slovenský ústav technickej normalizácie.

[5]. G.K.A. DIAS. International Journal of Computer Science and Software Engineering (IJCSSE), Volume 6, Issue 3, Marec 2017 ISSN (Online): 2409-4285.

[6]. PHAM HOANG. Software Reliability, s. 567, 1999, ISBN 9813083840.

[7]. AGRICULTURAL ECONOMICS : an international journal = mezinárodní vědecký časopis. Vol. 50, no. 2 (2004), s. 65-70. - Prague : Institute of Agricultural and Food Information, 2004. ISSN 0139-570X, Dostupné online: <http://www.agriculturejournals.cz/publicFiles/58690.pdf>.

[8]. JOHN P. VAN GIGCH. System Design Modeling and Metamodeling, Springer Science & Business Media, 2013, ISBN: 1489906762.

[9]. ISIS/VANDERBILT UNIVERSITY. WebGME Documentation, Release 1.0.0, Marec 2019, [cit, 2019-03-19], Dostupné online: https://media.readthedocs.org/pdf/webgme/latest/webgme.pdf.

[10]. RAHA NASERI. RPG modelling with Web GME, Department of Computer science, University of Antwerp, Január 2015, [cit: 2019-03-20], Dostupné online: <http://msdl.cs.mcgill.ca/people/hv/teaching/MSBDesign/201415/projects/Raha.Naseri/Final%20Report>.

[11]. M. MAROTI, T. KECSKES, R. KERESKENYI, P. VOLGYESI, A. LEDECZI. Online Collaborative Environment for Designing Complex Computational Sys15 tems,Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA., 2014, [cit: 2019-03-22], Dostupné online: https://webgme.org/WebGMEWhitePaper.pdf.

[12]. Á. LÉDECZI, M. MARÓTI, P. VÖLGYESI. The Generic Modeling Environment Technical Report; [cit: 2019-05-15], Dostupné online: <https://www.isis.vanderbilt.edu/sites/default/files/GMEReport.pdf >.

[13]. ISIS/VANDERBILT UNIVERSITY. GME: Generic Modeling Environment; [cit: 2019-05-16]; Dostupné online: < http://www.isis.vanderbilt.edu/projects/GMe >.

[14]. Sander Scrippier: Diagnosing verification and validation problems in public civil engineering projects, 2016, [cit: 2019-06-05]; Dostupné online: < https://pdfs.semanticscholar.org/4e43/d195ba1b0776da51d193b0bf95871188559d.pdf>.

[15]. IEEE-STD-610. A Compilation of IEEE Standard Computer Glossaries, IEEE Standard Computer Dictionary, 1991.

[16]. IEEE Standard Glossary of Software Engineering Terminology, September 1990, ISBN 1-55937-067-X, [cit: 2019-06-25], Dostupné online: < http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf>.

[17]. F. BUSCHMANN, R. MEUNIER, H. ROHNERT, P. SOMMERLAND, M. STAL. Pattern-Oriented Software Architecture Volume 1: A System of Patterns, 1996, ISBN: 978-0471958697, Dostupné online: <https://ff.tu-sofia.bg/~bogi/knigi/SE/Wiley%20-%20Pattern-Oriented%20Software%20Architecture%20-%20Volume%201,%20A%20System%20of%20Patterns.pdf>.

[18]. DOC. ING. ZDENĚK HAVLICE, CSC. Modelovanie a prototypovanie pri projektovaní informačných systémov, Január 1998, ISBN 80-88786-95-9.

[19]. HASSAN GOMAA. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures, 2011, ISBN 978-0-521-76414-8, [cit: 2020-03-30], Dostupné online: <https://books.google.sk/books?id=TqZi-hAX17YC>.

[20]. G. BOOCH, J. RUMBAUGH I. JACOBSON. Unified Modeling Language User Guide, Október 1998, ISBN: 0-201-57168-4, [cit: 2020-04-02], Dostupné online: < https://balu051989.files.wordpress.com/2011/06/the-unified-modeling-language-user-guide-by-grady-booch-james-rumbaugh-ivar-jacobson.pdf>.

[21]. STEVEN KELLY, JUHA-PEKKA TOLVANEN. Domain-Specific Modeling: Enabling Full Code Generation, November 2008, ISBN: 978-0-470-03666-2, [cit: 2020-04-08], Dostpné online: <https://books.google.sk/books?id=GFFtRFkuU_AC>.

[22]. ROBERT G. SARGENT. Verification and validation of simulation models, 2013, ISSN: 17477778, Dostupné online: <https://link.springer.com/article/10.1057%2Fjos.2012.20>.

[23]. M. LATUSZYNSKA. Problems of verification and validation of computer simulation models. STUDIA INFORMATICA. 27-40., Január 2013, Dostupné online: <

https://www.researchgate.net/publication/317722995_Problems_of_verification_and_validation_of_computer_simulation_models>.

[24]. SOFTWARE PRODUCTIVITY RESEARCH. SPR Programming Languages Table™ (PLT2005), http://www.spr.com, 2005.

[25]. STEVEN KELLY, PHD. Improving Developer Productivity With Domain-Specific Modeling Languages, Júl 2005, [cit: 2020-04-10], Dostupné online: <https://www.developerdotstar.com/mag/articles/domain_modeling_language.html>.

[26]. F. FERENČÍK. IGC2PDF Videoukážka, 2018, Dostupné online: < https://youtu.be/dP_jaGXk3vo>.

[27]. WEBGME: Wiki pages, 2016, [cit. 2020-04-25], Dostupné online: <https://github.com/webgme/webgme/wiki>.

Prílohy

Príloha A: CD médium – diplomová práca a prílohy v elektronickej podobe.

Príloha B: Systémová príručka

Príloha C: Metodická a používateľská príručka

Príloha D: Demonštrácia DSML na systéme igc2pdf

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SYSTÉMOVÁ PRÍRUČKA

Príloha B

2020 **František Ferenčík**

Obsah

Zoznam obrázkov

1. Súpis obsahu dodávky

CD priložené k diplomovej práci obsahuje:

- Zdrojové kódy aplikácie,
- systémovú príručku,
- metodickú a používateľskú príručku,
- demonštráciu DSML na systéme igc2pdf.

2. Inštalácia systému WebGME

Ak chceme začať s vývojom nových komponentov do systému WebGME, tak potrebujeme mať nainštalovanú vlastnú inštanciu WebGME servera.

2.1. Závislosti servera WebGME

Tento server má určité závislosti, ktoré je potrebné nainštalovať pred samotným zriadením WebGME servera. Tieto závislosti sú [1]:

- **Node.js** – minimálne verzia 6.
- **MongoDB** – minimálne verzia 3.
- **Git** - musí byť prístupný v premennej PATH.
- **Python** – nepovinné, iba ak budú rozšírenia vyvíjané v tomto jazyku.

2.2. Čistá inštalácia pomocou WebGME CLI

Vývojári WebGME pripravili užitočný nástroj WebGME CLI na správu WebGME inštalácií. Jeho hlavnou úlohou je vytváranie a mazanie inštalácií. Taktiež dôležitou funkciou je vytváranie rôznych komponentov systému WebGME, ktorými sú napríklad doplnok, dekoratér, vizualizér, rozšírenie, smerovač, rozloženie a iné.

Pred samotnou inštaláciou je potrebné mať nainštalované závislosti z predchádzajúcej kapitoly. Príkaz inštalácie WebGME CLI: **`npm install -g webgme-cli`**

Teraz je potrebné vytvoriť repozitár, v ktorom bude umiestnený nový server. Zároveň s ním sa vytvorí rovnomenný adresár. Repozitár sa vytvára pomocou príkazu: **`webgme init [nazov_repozitara]`**

Následne je potrebné sa presunúť do novovytvoreného adresára a nainštalovať všetky závislosti servera príkazom: **`npm install`**

Posledným krokom je spustenie servera, pričom je potrebné mať zapnutý databázový server MongoDB: **`mongod --fork --logpath [cesta_projektu]/log/mongod.log --dbpath [cesta_projektu]/webgmeData/`**

Server sa inicializuje príkazom: **`node app.js`** alebo **`npm start`**

Teraz po načítaní adresy *localhost:8888* vo webovom prehliadači sa zobrazí používateľské rozhranie WebGME. Prednastavený port je možné zmeniť v konfiguračnom súbore servera *config/config.webgme.js*.

S ukončením príkazového riadku sa ukončí aj proces webového servera. Preto odporúčame využiť CLI nástroj **forever**, ktorý tento proces spustí na pozadí.

Spustenie servera na pozadí: **forever start app.js**

Vypnutie servera na pozadí: **forever stop app.js**

2.3. Čistá inštalácia využitím Dockeru

Ďalším spôsobom inštalácie je spustenie Docker obrazu WebGME aplikácie v kontajneri. Výhodou tohto riešenia je, že Docker obraz je vytvorený so všetkými potrebnými závislosťami, takže nie je nutná ich manuálna inštalácia [2]. Vývojári WebGME pripravili obraz bežiacej aplikácie, ktorý je možné spustiť v kontajneri týmito príkazmi:

- Kontajner MongoDB - **`docker run -d -v ~/dockershare/db:/data/db --name mongo mongo`**
- Kontajner WebGME - **`docker run -d -p 8888:8888 -v ~/dockershare:/dockershare --link mongo:mongo --name=webgme webgme`**

Teraz vo webovom prehliadači na adrese *localhost:8888* je prístupné používateľské rozhranie WebGME.

2.4. Inštalácia WebGME spolu s vytvoreným CASE nástrojom

Inštalačný balík je umiestnený v prílohe A, ktorá obsahuje zdrojové kódy s nakonfigurovaným CASE nástrojom MVCStudio [3]. Je potrebné si skopírovať projekt na zodpovedajúce miesto, na ktorom bude bežať webový server. Následne je potrebné stiahnuť všetky NODE závislosti pomocou príkazu: **`npm install`**.

V konfigurácii *config/config.webgme.js* je potrebné nastaviť spojenie s databázovým serverom MongoDB.

Teraz je potrebné spustiť server pomocou jednej z metód popísaných v kapitole 2.2.

Na záver po spustení WebGME vo webovom prehliadači je potrebné importovať projekt spolu s meta-modelmi, ktoré sú vyexportované v prílohe A v súbore *MVCStudio.webgmex.*

3. Inštalácia a konfigurácia obslužného programu pre vývojové prostredia

Obsahom tejto kapitoly návod na inštaláciu a nastavenie obslužného programu pre otváranie zdrojových kódov vo vývojovom prostredí.

3.1. Popis obslužného programu

Tento program slúži na otvorenie reprezentácie artefaktu modelu v zdrojovom kóde vo vybranom vývojovom prostredí IDE na klientskom počítači. Tento návod je určený pre systémy s operačným systémom typu Microsoft Windows.

Nástroj funguje na princípe špecifického, pre tento účel vytvoreného protokolu *editor://*, ktorý obsahuje identifikátory zdrojového kódu, ktorý chceme otvoriť vo vývojovom prostredí. Tento protokol sa vyvolá stlačením na tlačidlo s hypertextovým odkazom obsahujúci spomínaný protokol. Tento protokol odchytí systém Windows a posunie obslužnému programu, ktorý na základe konfigurácie vyberie vývojové prostredie a posunie mu zdrojový kód na otvorenie z obdržaného protokolu.

Protokol vyzerá nasledovne: `editor://open/?file=APP_ROOT\app/Template.php&line=1`

Tento nástroj dokáže otvoriť zdrojový kód s kurzorom na definovanom riadku, avšak táto funkcia nie je v tomto nástroji plne využitá (nastavený je prvý riadok), no pri budúcich rozšíreniach je možné prostredníctvom tejto funkcie nasmerovať programátora na určitú metódu alebo dokonca na určitú problematickú konštrukciu zdrojového kódu, ktorá môže byť predmetom jeho záujmu.

Riešenie je pripravené pre väčšinu moderných vývojových prostredí a editorov zdrojových kódov používaných pre vývoj webových aplikácií, pričom je jednoduché nakonfigurovať podporu aj na ďalšie vývojové prostredia. Aktuálne podporované editory a vývojové prostredia sú:

- PhpStorm,
- NetBeans,
- Nusphere PHPEd,
- SciTE,
- EmEditor,
- PSPad Editor,
- gVim,
- Sublime Text 3.

3.2. Vytvorenie obslužného programu

Obslužný program je napísaný v programovacom jazyku JavaScript.

- Vytvorte adresár na disku podľa cesty **C:\ProgramFiles\EditorProtocolHandler;**
- Vytvorte súbor **editor.js**;
- Nakopírujte do neho nasledujúci zdrojový kód.

```
var settings = {
        // PhpStorm
        //  editor:  '"C:\\Program  Files\\JetBrains\\PhpStorm  2019.1\\bin\\phpstorm64.exe"  --line
%line% "%file%"',
        // title: 'PhpStorm',
        // NetBeans
        // editor: '"C:\\Program Files\\NetBeans 8.1\\bin\\netbeans.exe" "%file%:%line%" --console
suppress',
        // Nusphere PHPEd
        // editor: '"C:\\Program Files\\NuSphere\\PhpED\\phped.exe" "%file%" --line=%line%',
        // SciTE
        // editor: '"C:\\Program Files\\SciTE\\scite.exe" "-open:%file%" -goto:%line%',
        // EmEditor
        // editor: '"C:\\Program Files\\EmEditor\\EmEditor.exe" "%file%" /l %line%',
        // PSPad Editor
        // editor: '"C:\\Program Files\\PSPad editor\\PSPad.exe" -%line% "%file%"',
        // gVim
        // editor: '"C:\\Program Files\\Vim\\vim73\\gvim.exe" "%file%" +%line%',
        // Sublime Text 3
        //editor: '"C:\\Program Files (x86)\\Sublime Text 3\\sublime_text.exe" "%file%:%line%"',
        mappings: {
                'APP_ROOT': 'C:\\Program Files (x86)\\Ampps\\www\\igc2pdf\\'
        }
};
if (!settings.editor) {
        WScript.Echo('Create variable "settings.editor" in ' + WScript.ScriptFullName);
        WScript.Quit();
}
var url = WScript.Arguments(0);
var                                             match                                                   =
/^editor:\/\/(open|create|fix)\/\?file=([^&]+)&line=(\d+)(?:&search=([^&]*)&replace=([^&]*))?/.exec
(url);
if (!match) {
        WScript.Echo('Unexpected URI ' + url);
        WScript.Quit();
}
for (var i in match) {
        match[i] = decodeURIComponent(match[i]).replace(/\+/g, ' ');
}
var action = match[1];
var file = match[2];
var line = match[3];
var search = match[4];
var replace = match[5];
var shell = new ActiveXObject('WScript.Shell');
var fileSystem = new ActiveXObject('Scripting.FileSystemObject');
for (var id in settings.mappings) {
        if (file.indexOf(id) === 0) {
                file = settings.mappings[id] + file.substr(id.length);
                break;
        }
}
if (action === 'create' && !fileSystem.FileExists(file)) {
        shell.Run('cmd /c mkdir "' + fileSystem.GetParentFolderName(file) + '"', 0, 1);
        fileSystem.CreateTextFile(file);
} else if (action === 'fix') {
        var lines = fileSystem.OpenTextFile(file).ReadAll().split('\n');
        lines[line-1] = lines[line-1].replace(search, replace);
        fileSystem.OpenTextFile(file, 2).Write(lines.join('\n'));
}
var command = settings.editor.replace(/%line%/, line).replace(/%file%/, file);
```

```
shell.Exec(command);
if (settings.title) {
        shell.AppActivate(settings.title);
}
```

3.3. Konfigurácia obslužného programu

V štruktúre *settings* sa nachádzajú všetky potrebné nastavenia. Pre výber želaného vývojového prostredia je potrebné odstrániť komentár príslušného riadku a nastaviť cestu k jeho spúšťaciemu súboru, pretože tieto cesty sa líšia v závislosti od verzie operačného systému a taktiež od verzie vývojového prostredia.

Štruktúra *settings* taktiež obsahuje štruktúru *mappings*, kde je potrebné zadať cestu ku koreňovému priečinku zdrojových kódov cieľového systému.

3.4. Inštalácia obslužného programu

Posledným krokom je nakonfigurovanie systému Windows, aby odchytený protokol *editor://* poslal na spracovanie nášmu obslužnému programu:

- Vytvorte súbor *install.cmd;*
- Nakopírujte do neho nasledujúci skript;
- Spustite tento súbor s administrátorskými oprávneniami.

```
@echo off
if defined PROCESSOR_ARCHITEW6432 (set reg="%systemroot%\sysnative\reg.exe") else (set
reg=reg)
%reg% ADD HKCR\editor /ve /d "URL:editor Protocol" /f
%reg% ADD HKCR\editor /v "URL Protocol" /d "" /f
%reg%  ADD  HKCR\editor\shell\open\command  /ve  /d  "wscript  \"%~dp0open-editor.js\"
\"%%1\"" /f
```

4. Rozšírenia systému CASE nástroja

Po vytvorení CASE nástroja na platforme WebGME s implementovaným DSML boli implementované rozšírenia. Pomocou nich sme sa pokúsili niektoré činnosti automatizovať, uľahčiť a zefektívniť.

Prostredie WebGME je vyvinuté s dôrazom na použitie s rozsiahlymi modelmi a teda nie sú vždy dostupné všetky informácie o všetkých modeloch v rámci projektu. Načítavajú sa len nevyhnutné časti modelov pre kontext, v ktorom sa používateľ nachádza. Takýto prístup vedie k asynchrónnemu programovaniu, pretože ak chceme pracovať s dátami, ktoré aktuálne nemáme načítané, tak musíme o nich požiadať jadro systému prostredníctvom asynchrónnych volaní pomocou JavaScript mechanizmu *Promise*.

Aj keď je k dispozícii aj programovací jazyk Python, zvolili sme JavaScript z dôvodu, že JavaScript rozšírenia môžu bežať ako na strane servera, tak aj na strane klienta.

4.1. Štruktúra projektu

Platforma WebGME prichádza s určitou zadefinovanou štruktúrou projektu, ktorá je pre vývojárov záväzná. Pre stručnosť uvedieme iba jej prvky využité v rámci tejto práce:

- blob-local-storage/ – úložisko vygenerovaných artefaktov pluginmi,
- config/ - adresár s nastaveniami,
  - confiig.default.js – hlavná konfigurácia rozšírení,
  - config.webme.js – nastavenia jadra systému,
- node_modules/ - adresár s nainštalovanými závislosťami,
- src/ - hlavný adresár pre zdrojové kódy rozšírení,
  - client/... - súbory prístupné pre webový prehliadač,
  - common/... - znovu použiteľné kódy zdieľané medzi rozšíreniami,
  - decorators/... - dekoratéry,
  - plugins/... - rozšírenia,
  - visualizers/... – vizualizéry,
- app.js – spúšťací súbor,
- package.json – definícia JavaScript závislostí.

4.2. Rozšírenia – pluginy

V tejto kapitole sú popísané rozšírenia (pluginy). Rozšírenie je možné povoliť pre určitý model pomocou jeho povolenia vo WebGME editore v sekcii: *Property editor -> Meta -> validPlugins.*

4.2.1. Vytvorenie rozšírenia

V kapitole 2.2 už bol spomenutý nástroj WebGME CLI. Tento nástroj vytvorí nové rozšírenie s definovaným menom a zaregistruje ho do systému. Následne je potrebné webový server aplikácie reštartovať, aby sa načítala zmenená konfigurácia.

Použitie: **`webgme-cli new [typ] [nazov]`**

Typ môže byť *addon, decorator, layout, plugin, router, seed, viz.*

4.2.2. ArtifactExporter

Jeho účelom je analyzovať všetky prvky vytvoreného modelu systému a vyexportovať dôležité informácie nielen pre potreby generovania zdrojových kódov vo forme JSON reťazca. Po úspešnom vykonaní sa v správe o výsledku nachádza vygenerovaný JSON súbor. Toto rozšírenie môže byť inicializované aj iným rozšírením, pričom preberie jeho výstup a ďalej s ním pracuje.

Jeho umiestnenie v projekte je src/plugins/ArtifactExporter/ArtifactExporter.js

Princípom je vyžiadanie si z jadra WebGME všetkých potomkov aktuálneho uzlu a po skontrolovaní ich meta-typu zaradenie do príslušného zoznamu. Následne sa z kompletných zoznamov artefaktov generuje JSON reťazec, ktorý je uložený do *blob* úložiska a taktiež je pripojený k výsledným návratovým dátam pre používateľa vo forme súboru *source-structure.json.*

V prípade extrémne veľkých modelov, ktoré by spôsobovali vyčerpanie pamäťových zdrojov, by bolo možné si vyžiadať z jadra iba zoznam identifikátorov artefaktov. Cez tie sa budú iterovať a sťahovať jednotlivé modely a spracovávať po častiach.

Skrátený príklad výsledného súboru je zobrazený na Obr. 1.

Obr. 1: Skrátený vzor výstupu pluginu ArtifactExporter

4.2.3.SourceCodeGenerator

Generovanie zdrojových kódov má na starosti toto rozšírenie. Pred spustením v kontextovom okne Obr. 2 je potrebné nastaviť metódu generovania zdrojových kódov a zapnúť generovanie databázových migračných súborov pre modely. Po spustení sa automaticky inicializuje rozšírenie *ArtifactExporter*, od ktorého si preberie schému vytvorených artefaktov a vygeneruje z nej *kontroléry* a *modely,* prípadne aj databázové migračné súbory. Zdrojové kódy budú umiestnené na základe definovaných menných priestorov v rámci konkrétneho modelu do vývojovej verzie projektu softvérového systému, ktorá je definovaná v ROOT koreňovom uzli modelu projektu.

Umiestnenie pluginu v projekte: webgme-app/src/plugins/SourceCodeGenerator/ SourceCodeGenerator.js

Obr. 2: Konfigurácia rozšírenia *SourceCodeGenerator*

4.2.4.ArtifactGenerator

Toto rozšírenie dokáže zo zoznamu požiadaviek automaticky vygenerovať predbežný návrh nových chýbajúcich systémových objektov. Požiadavky sú spísané v textovej forme v anglickom jazyku a sú vložené do uzla *Requirements*. Plugin na základe slovno-druhovej analýzy extrahuje potencionálne nové artefakty. Slovno-druhová analýza je vykonávaná pomocou JavaScript knižnice s názvom *Compromise.* Extrahované slová sú ďalej upravené a na základe podmienok uvedených v Tab. 1 sú vložené do modelu ako nové artefakty, ak ešte neexistujú.

Umiestnenie pluginu v projekte: webgme-app/src/plugins/ArtifactGenerator/ ArtifactGenerator.js

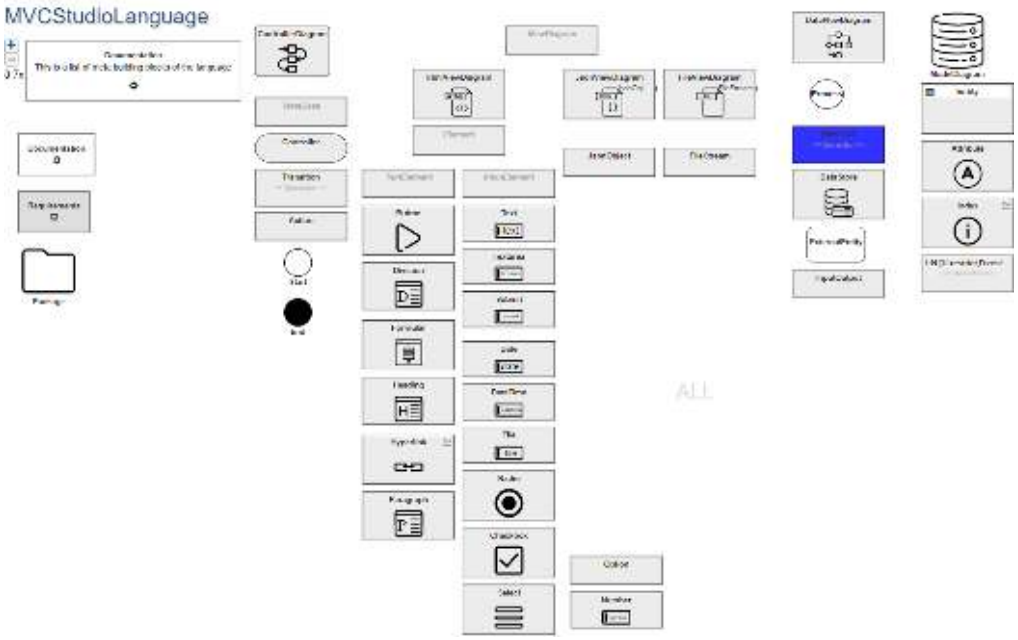
Tab. 1: Mapovanie generovania artefaktov zo slovných druhov pre modely

Model	Podstatné meno	Sloveso
Package	ControllerDiagram	-
ModelDiagram	Entity	-
ControllerDiagram	Controller	-
Controller	-	Action
DataFlowDiagram	DataStore	Process
HtmlViewDiagram	Division	-
Formular	Text	-
JsonViewDiagram	JsonObject	-
JsonObject	JsonObject	-
Entity	Attribute	-

4.3. Dekoratéry

Pre špecifickú vizualizáciu jednotlivých uzlov v modeloch boli implementované viaceré dekoratéry zobrazené na Obr. 3:

- BasicDecorator – všeobecný dekoretér pre pridanie prepojenia na zdrojový kód.
- ControllerDecorator – dekorácia kontroléra.
- ControllerDiagramDecorator – dekorácia uzla pre diagram kontrolérov.
- EntityDecorator – dekorácia uzlu databázová entita.
- ExternalEntityDecorator – dekorácia uzla externá entita.
- LanguageDecorator – dekorácia uzla pre jazyk DSML.
- PackageDecorator – dekorácia balíka.



Obr. 3: Prvky DSML jazyka

4.3.1.BasicDecorator

Účelom tohto rozšírenia je jedným klikom sa dostať od modelu do implementácie artefaktu v podobe zdrojového kódu. Táto funkcia bola implementovaná do artefaktov CASE nástroja MVCStudio v podobe tlačidiel v jeho spodnej časti uzla modelu viď. Obr. 4.

Obr. 4: Tlačidlá na zdrojové kódy

Zdrojové kódy je možné otvoriť dvoma spôsobmi:

- **git** - po kliknutí na tlačidlo ***git*** sa vo webovom prehliadači otvorí nová karta zobrazujúca zdrojový kód aktuálneho artefaktu vo webovom rozhraní systému pre správu verzií *gitlab*;
- **vývojové prostredie** – po kliknutí na piktogram **</>** sa otvorí zdrojový kód artefaktu vo vývojovom prostredí na používateľovom počítači.

Popis tohto rozšírenia je obsahom kapitoly 3.1. Pred prvým použitím je však potrebné nainštalovať a nakonfigurovať obslužný program hostiteľského počítača. Návod s postupom pri tejto inštalácii je umiestnený v kapitole 3.

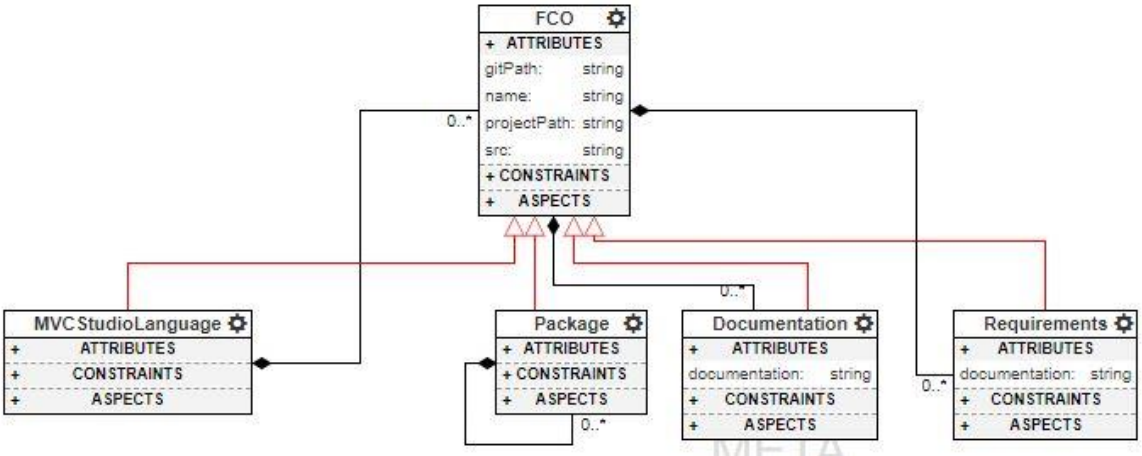
5. Implementácia metodológie do CASE nástroja MVCStudio pomocou DSML meta-modelovania

Obsahom tejto kapitoly je vytvorenie meta-modelov z DSML pre metodológiu modelovania webových systémov založených na architektúre MVC.

Celkový meta-model systému bol pre zachovanie prehľadnosti rozbitý do viacerých malých ucelených meta-modelov. Keďže jednotlivé meta-modely sú medzi sebou prepojené, tak objekty, ktoré patria určitému meta-modelu, sú farebne zvýraznené rovnakou farbou. To znamená, že keď v meta-modeli je objekt inej farby, tak pochádza z iného meta-modelu a je sem pridaný za účelom prepojenia týchto meta-modelov.

5.1. Základný meta-model

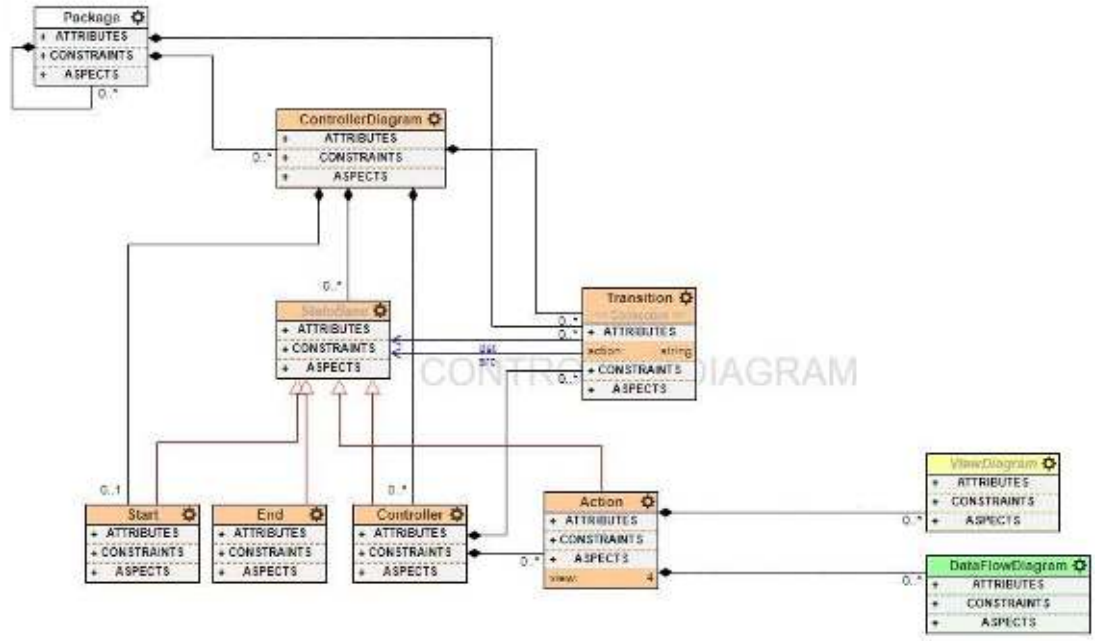
Tento meta-model je znázornený na Obr. 5. Obsahuje objekt *FCO – first class object,* ktorý tvorí koreň stromu dedičnosti. Ďalej je tu objekt *MVCStudioLanguage*, ktorý v kompozícii kvôli prehľadnosti združuje všetky objekty modelovacieho jazyka. Následne je tu objekt *Documentation*, ktorý umožňuje vytvárať dokumentáciu v ľubovoľnom kontexte pre ľubovoľný model, diagram alebo jeho uzol, čo zabezpečuje prepojenie s uzlom *FCO* s kardinalitou 0:N. Objekt *Requirements* umožňuje zaznamenávať textové požiadavky v ľubovoľnom kontexte rovnako ako *Documentation*. Posledným objektom je tu objekt *Package,* ktorý umožňuje lepšie štrukturovať projekt pomocou balíčkov, ktoré môžu byť vnorené, a teda balíček môže obsahovať ďalší balíček a tým tvoriť stromovú hierarchiu zlučujúcu navzájom súvisiace modely.



Obr. 5: Základný meta-model

5.2. Meta-model ControllerDiagram

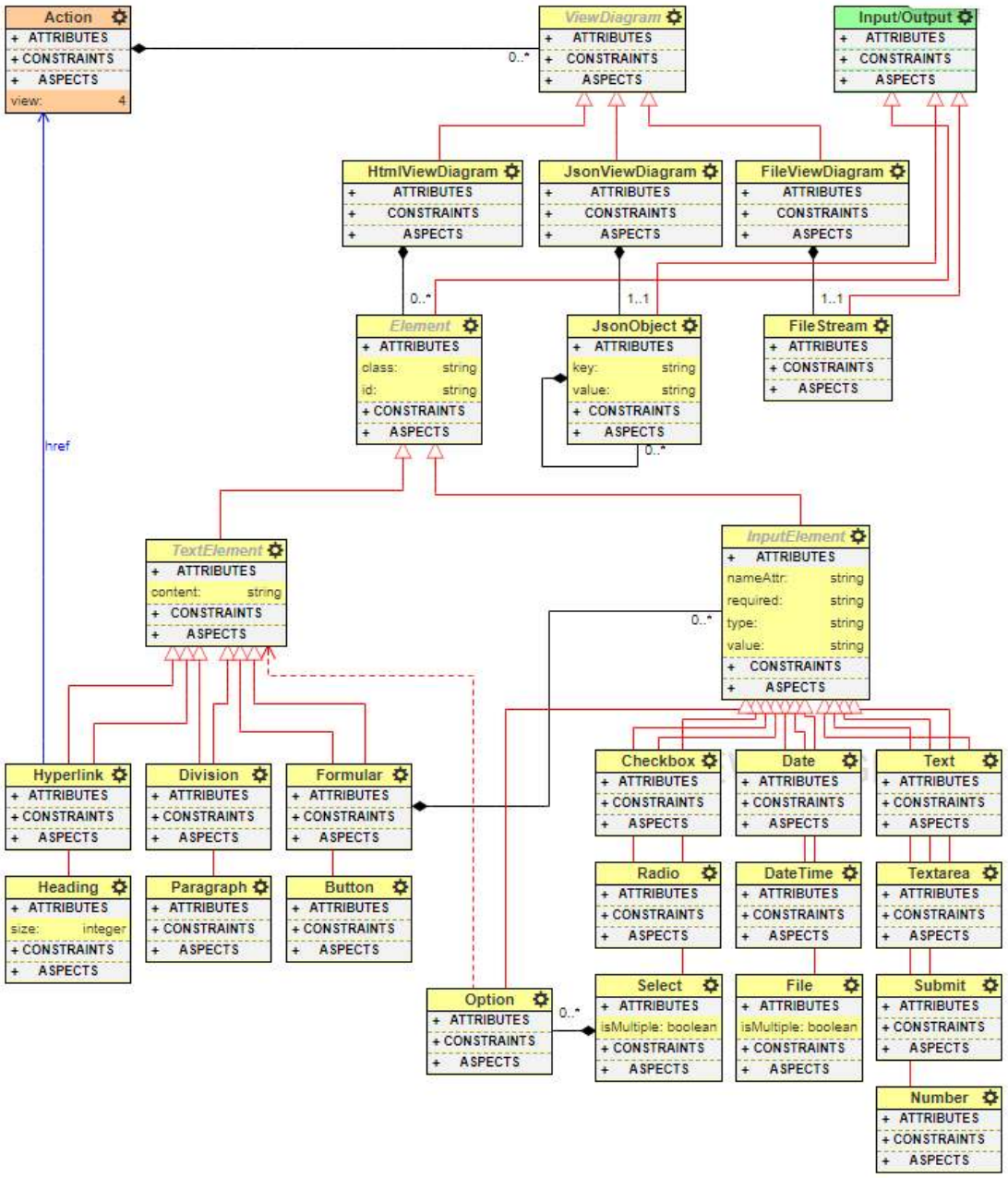
Na Obr. 6 je zobrazený meta-model pre *ControllerDiagram,* pomocou ktorého sa modelujú rozličné stavy aplikácie. Ten obsahuje objekt *ControllerDiagram* reprezentujúci samotný diagram. Je prepojený s *Package* čo znamená, že balíček môže obsahovať viacero týchto diagramov. Ďalej sú s ním prepojené objekty, ktoré tento diagram obsahuje, ktorými sú *Controller, Action, Start, End.* Tieto objekty dedia z objektu *StateBase,* ktorý je prepojený s prepojovacím objektom *Transition*. Toto prepojenie teda zdedia všetky komponenty, čo umožňuje ich vzájomné prepájanie. Nachádza sa tu taktiež prepojenie na ďalšie diagramy. To umožňuje do objektu *Action* vytvoriť viaceré objekty *ViewDiagram* a *DataFlowDiagram.*



Obr. 6: Meta-model ControllerDiagram

5.3. Meta-model View diagram

Na Obr. 7 je znázornený meta-model *View* diagramu pre modelovanie pohľadu a teda aj používateľského rozhrania. Ten obsahuje abstraktný objekt *ViewDiagram* zlučujúci všetky komponenty diagramu. Od neho dedia všetky konkrétne druhy *View* diagramov. Prvým je *FileViewDiagram,* ktorý môže obsahovať *FileStream*, potom je tu *JsonViewDiagram,* ktorý obsahuje *JsonObject* a posledným najvyužívanejším je *HtmlViewDiagram,* ktorý obsahuje objekty typu *Element*, ktoré sa ďalej špecifikujú v rámci hierarchie dedičnosti na konkrétne HTML komponenty. Čo sa týka prepojení, tak objekt typu *Hyperlink* má prepojenie s *Action,* čo umožňuje vytvárať hypertextové prepojenia medzi odkazom a akciou (*Action* z diagramu *Controller*). Taktiež je tu objekt typu *Input/Output* z meta-modelu diagramu *DataFlow,* od ktorého dedia určité objekty. To umožňuje prepojiť *DataFlow* diagram s *View* diagramom.

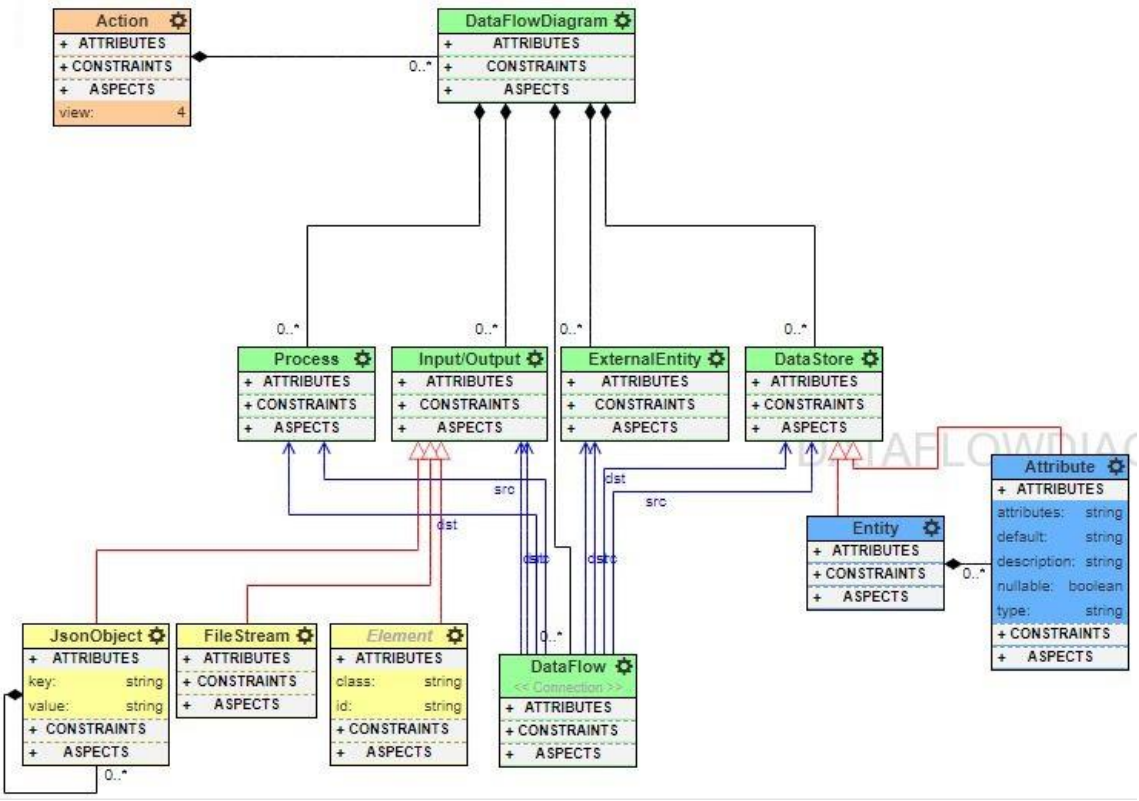


Obr. 7: Meta-model *View* diagramu

5.4. Meta-model DataFlowDiagram

Tento meta-model zachytený na Obr. 8 obsahuje *DataFlowDiagram* modelujúci dátový tok aplikácie. Obsahuje komponenty ako je *Proces, Input/Output, ExternalEntity, DataStore*. Tieto je možné v modeli prepájať vďaka objektu *DataFlow*, ktorý má s nimi nastavené ukazovatele. Čo sa týka prepojení, tak *DataFlowDiagram* je prepojený s *Action,* čo umožňuje prepojiť *DataFlow* diagram s *Controller* diagramom. Ďalej od objektu *DataStore* dedia objekty *Entity a Attribute,* čo umožňuje prepojenie *DataFlow* diagramu s *Model* diagramom. Nakoniec sú tu žlté objekty z *View*

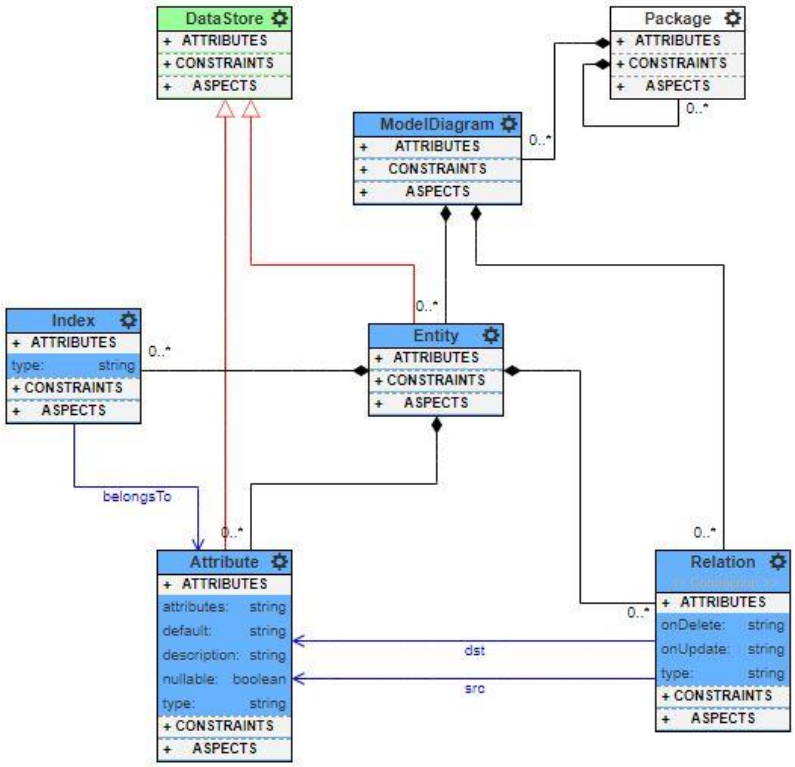
diagramu, ktoré dedia od objektu *Input/Output*, čo zabezpečuje prepojenie *DataFlow* diagramu s *View* diagramom.



Obr. 8: Meta-model DataFlow diagramu

5.5. Meta-model Model diagram

Posledným meta-modelom je na Obr. 9 znázornený Model diagram reprezentujúci dátovú vrstvu aplikácie. Ten obsahuje objekt *ModelDiagram*, ktorý je prepojený s objektom *Package*, čiže balíček projektu môže obsahovať tento diagram. Ďalej obsahuje objekty *Entity a Attribute*. Objekt *Index* je prepojený s *Attribute*, čo umožňuje definovať indexy pre určitý stĺpec databázy. Objekt *Attribute* je taktiež prepojený s objektom prepojenia *Relation,* čo umožňuje vytvárať prepojenia vo forme cudzích kľúčov. Nachádza sa tu aj objekt *DataStore* z *DataFlowDiagram* meta-modelu. Od neho dedia *Entity a Attribute*, čo umožňuje v DataFlow diagrame prepájať procesy s databázovými entitami a taktiež aj s jej atribútmi.



Obr. 9: Meta-model Model diagramu

Zdroje

[1]. ISIS/VANDERBILT UNIVERSITY: How to build a Design Studio with WebGME, 2017, [cit. 2020-04-25], Dostupné online: <https://webgme.readthedocs.io/en/latest/index.html>.

[2]. WEBGME: Wiki pages, 2016, [cit. 2020-04-25], Dostupné online: <https://github.com/webgme/webgme/wiki>.

[3]. FRANTIŠEK FERENČÍK: Metamodelovanie, modelovanie, verifikácia a validácia požiadaviek. Diplomová práca, Technická univerzita v Košiciach, 2020.

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

METODICKÁ A POUŽÍVATEĽSKÁ PRÍRUČKA

Príloha C

2020 **František Ferenčík**

Obsah

Zoznam obrázkov

1. Súpis obsahu dodávky

CD priložené k diplomovej práci obsahuje:

- Zdrojové kódy aplikácie,
- systémovú príručku,
- metodickú a používateľskú príručku,
- demonštráciu DSML na systéme igc2pdf.

2. Úvod

Táto príručka sa zameriava na CASE nástroj MVCStudio založený na platforme WebGME.

Tento nástroj bol vytvorený v rámci diplomovej práce [1] a je určený na podporu činností počas SwLC na systémoch špeciálnej kategórie, ktorými sú webové aplikácie postavené na MVC architektúre. Modelovanie prebieha prostredníctvom pre tento účel navrhnutého DSML.

Príručka plní dvojakú úlohu, pričom prvou z nich je definovanie metodiky, ako postupovať pri návrhu a modelovaní systémov pomocou DSML. Druhou úlohou je návod pre používateľov, ako samotný CASE nástroj použiť.

Pre správne použitie nástroja tieto dve úlohy sú od seba neodlučiteľné, a z toho dôvodu budú popisované súbežne.

Predpoklady pre použitie tohto nástroja sú:

- Nainštalovaný CASE nástroj MVCStudio. Návod na jeho inštaláciu a konfiguráciu sa nachádza v systémovej príručke.
- Ovládať návrhový vzor MVC.
- Ovládať princípy fungovania webových aplikácií.
- Skúsenosti s vývojom vo webovom systémovom rámci (napr. Laravel, Symfony, Zend, Nette, atď.) sú veľkým benefitom, keďže väčšina moderných rámcov využíva práve MVC architektúru.

3. Postup pri návrhu a modelovaní systému

Počas návrhu a modelovania systému sa potrebujeme zamerať na jeho kritické časti. V tomto prípade tieto časti sú dátová vrstva, riadenie stavov, GUI a dátový tok.

Neexistuje striktný univerzálny postup, v akom poradí je potrebné takýto systém navrhovať a zostavovať jednotlivé diagramy, avšak odporúčame sa inšpirovať naším odporúčaným postupom a jednotlivé kroky prispôsobovať pre potreby konkrétnej situácie:

1. Príprava a konfigurácia projektu. Postup je uvedený v kapitole 4.
2. Na základe definícií požiadaviek na systém je potrebné sa zamyslieť nad dátovou vrstvou a pokúsiť sa ju namodelovať. Ak sa v prvotnom návrhu budú objavovať určité chyby a nezrovnalosti, ničomu to nevadí, pretože kedykoľvek je možné tieto modely upraviť. Ak na začiatku ešte nie je jasná štruktúra dát, je potrebné namodelovať aspoň to, čo je známe, a ostatné elementy budú doplnené priebežne počas návrhu a zostavovania iných modelov. Postup modelovania dátovej vrstvy je uvedený v kapitole 5.
3. Z definícií požiadaviek na systém je potrebné odhaliť hlavné funkcie systému. Tieto funkcie je potrebné rozdeliť do navzájom súvisiacich skupín. Zvyčajne sa rozdeľujú najprv na väčšie funkčné celky, ktoré sa nazývajú moduly (administrácia, profil používateľa, predný modul, atď.). Postup sa nachádza v kapitole 6.
4. Následne pre každý modul znova skúmame funkcie systému a opäť sa ich snažíme rozdeliť do vzájomne súvisiacich skupín. Tieto skupiny môžeme chápať ako stavy, v ktorých sa používateľ môže nachádzať (sekcia importu záznamov, výber šablóny, generovanie PDF, …). V rámci týchto stavov môžu byť definované aj podstavy, ktoré označujeme ako akcie. Ich význam je v podpore čiastkových funkcií, ktoré poskytuje daný kontrolér (nahranie súboru). Postup pri modelovaní stavov sa nachádza v kapitole 7.
5. Ďalším krokom je navrhnutie a namodelovanie GUI a iné typy pohľadov (HTML šablóna, súbor, JSON string). Tieto pohľady obsahujú pre daný typ špecifické komponenty. Z týchto komponentov vyskladáme požadovaný pohľad. Teda napríklad v prípade modelovaní GUI šablóny modelujeme formuláre, vstupné polia, odkazy, tlačidlá a veľa ďalších elementov. Postup sa nachádza v kapitole 8.
6. V tomto momente máme k dispozícii a namodelované všetko, čo potrebujeme pre zostavenie diagramu toku údajov. Je vhodné mať už zostavené predchádzajúce diagramy, pretože sú využívané vo veľkej miere. Preto odporúčame tento typ diagramu modelovať úplne na záver. Dátový tok dokáže prepájať dátovú vrstvu s pohľadovou vrstvou prostredníctvom procesov v kontroléroch. Návod sa nachádza v kapitole 9.

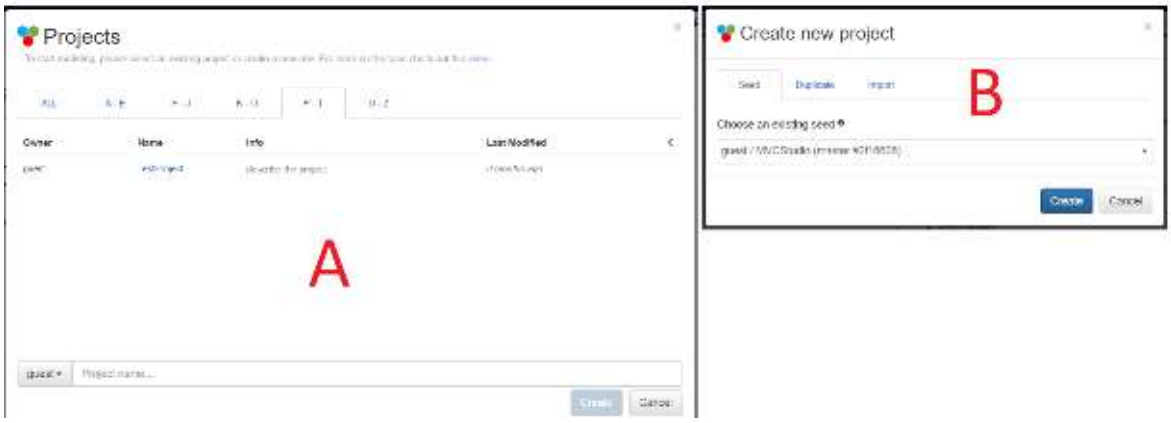
4. Prvé kroky – príprava projektu

Táto kapitola popisuje kroky súvisiace s prípravou projektu, ktoré je potrebné urobiť na začiatku každého projektu.

4.1. Vytvorenie projektu

Pre vytvorenie projektu sú potrebné nasledujúce kroky:

1. Načítaním systému WebGME vo webovom prehliadači sa zobrazí okno *Projects* (Obr. 1 – A). Tu sa zobrazujú všetky projekty, pričom po kliknutí na názov sa projekt otvorí.
2. Stlačiť na tlačidlo *Create new...,* vyplniť názov projektu a potvrdiť stlačením *Create*.
3. Zobrazí sa ďalšie okno (Obr. 1 - B), v ktorom je možné vytvoriť projekt 3 spôsobmi, a to seedovaním, duplikovaním projektu MVCStudio alebo importom *webgmex* súboru, ktorý je súčasťou dodania v elektronickej prílohe.



Obr. 1: Vytvorenie projektu

4.2. Konfigurácia projektu

Teraz je možné urobiť základné nastavenia projektu:

1. Stlačiť na uzol *FCO.*
    a. Na pravej spodnej časti editoru v časti *Property editor* prepnúť do *Attributes.*
    b. Vyplniť gitPath – URL ku koreňovému adresáru projektu v gitlabe.
    c. projectPath – nastaviť cestu k vývojovej verzii projektu.
2. Vytvoriť balík projektu, ktorý bude združovať všetky modely navrhovaného systému.
    a. Dvojkliknutím na uzol *Projects* ho otvoríme a dostaneme sa do jeho kontextu.
    b. Zo zoznamu dostupných artefaktov (Obr. 2 - A) vyberieme *Package* a pretiahnutím do editora vytvoríme jeho inštanciu.
    c. Nastavíme jeho názov v *Property editor -> Attributes -> name.*

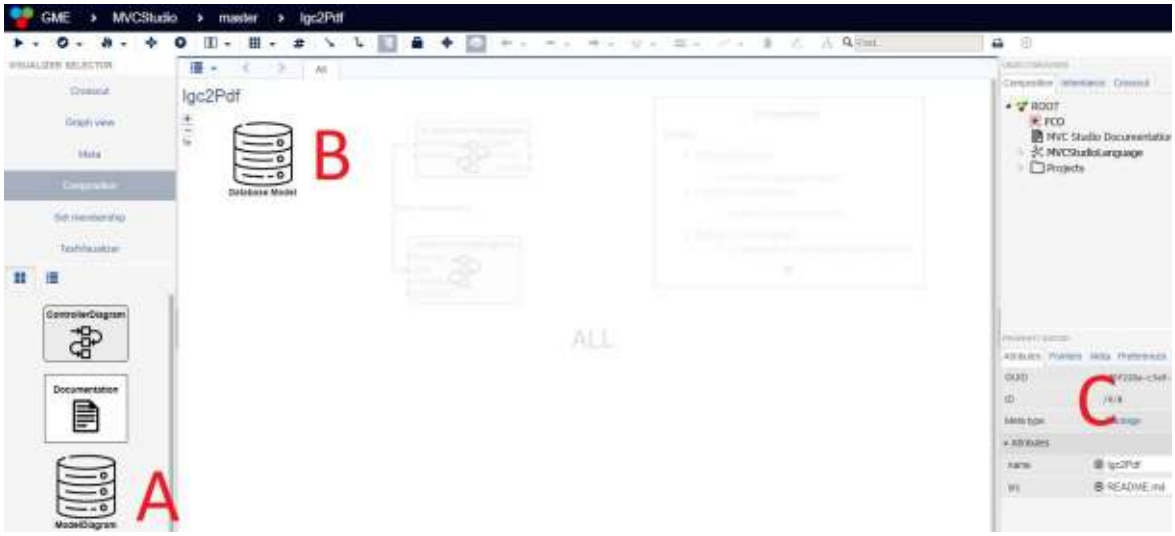
5. Modelovanie dátovej vrstvy

Modelujeme ju pomocou diagramu s názvom *ModelDiagram*.

5.1. Vytvorenie diagramu *ModelDiagram*

Postup pri vytvorení je nasledovný:

1. Prepneme sa do balíka projektu, pre ktorý tento diagram chceme vytvoriť.
2. Z dostupných artefaktov (Obr. 2 - A) pretiahneme do editora *ModelDiagram* (Obr. 2 - B).
3. Nastavíme mu požadované meno v *Property editor* (Obr. 2 - C).
4. Otvoríme diagram.



Obr. 2: Vytvorenie diagramu *ModelDiagram*

5.2. Vytvorenie modelových entít

Jednotlivé databázové tabuľky sú v aplikácii reprezentované svojimi entitami. Postup pri ich vytváraní je nasledovný:

1. Zo zoznamu dostupných artefaktov vyberieme *Entity* a vytvoríme jeho inštanciu v editore.
2. Nastavíme názov entity v *Property editor.*
3. Môžeme nastaviť jeho menný priestor spolu s názvom v *Property editor.*

5.3. Vytvorenie atribútov a indexov entity

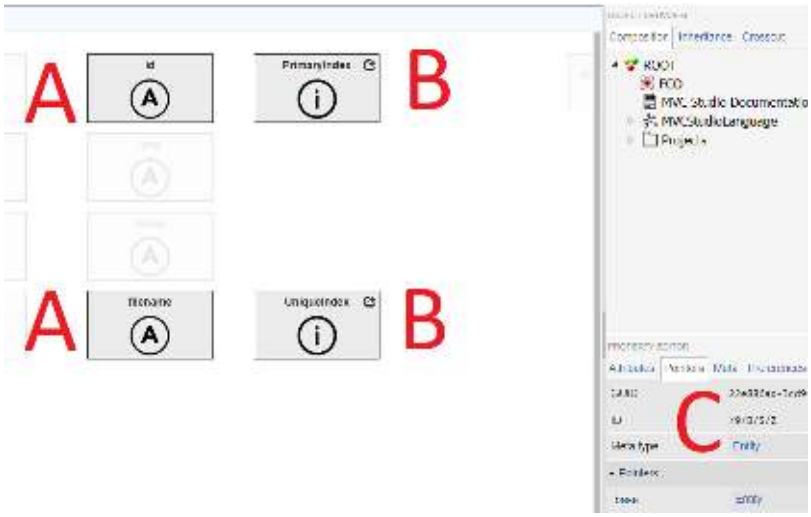
Databázové tabuľky obsahujú určité stĺpce, ktoré modelujeme pomocou uzla *Attribute* a taktiež obsahujú aj určité indexy vzťahujúce sa na tieto atribúty. Tie modelujeme pomocou uzla *Index.* Pri modelovaní atribútov (Obr. 3 - A) postupujeme nasledovne:

1. Zo zoznamu artefaktov vytvoríme inštanciu *Attribute.*

2. Nastavíme atribúty v *Property editor:*
    a. attributes - unsigned / binary ...,
    b. default – prednastavená hodnota,
    c. description – popis
    d. name – názov
    e. nullable – povoliť prázdne hodnoty NULL,
    f. type – dátový typ napr. INT(11), VARCHAR(255) atď.
3. Opakovať postup pre všetky stĺpce databázovej tabuľky.

Pri modelovaní indexov (Obr. 3 - B) postupujeme nasledovne:

1. Zo zoznamu artefaktov vytvoríme inštanciu objektu *Index.*
2. Nastavíme atribúty v *Property editor:*
    a. *name – názov,*
    b. *type – typ indexu* – index / primárny kľúč / unikátna hodnota,
3. Index sa vzťahuje na atribút, ktorý je potrebné definovať jeho presunutím nad index v editore. S prepojením je možné pracovať v (Obr. 3 - C) *Property editor -> Pointers.*
4. Opakovať postup pre všetky indexy databázovej tabuľky.



Obr. 3: Modelovanie atribútov a indexov

5.4. Modelovanie prepojení entít s obmedzeniami zabezpečenia RI

V relačných databázach sú medzi tabuľkami vytvorené určité relácie, ktoré budeme modelovať orientovanými hranami vo forme šípok prepájajúcich jednotlivé atribúty entít. Relácie môžu mať rôzne druhy kardinality a môžu obsahovať určité pravidlá pre zabezpečenie referenčnej integrity v prípade zmeny alebo vymazania kľúčového záznamu.

Postup pri modelovaní prepojenia entít:

1. Prejdením myšou nad entitou sa nad portmi jej atribútov zobrazia štvorce.
2. Na tento štvorec klikneme, držíme a ťaháme na štvorec atribútu v druhej entite.
3. Vytvorí sa orientovaná hrana od potomka ku kľúču (Obr. 4 - A).
4. Je potrebné nastaviť v (Obr. 4 - B) *Property editor:*
    a. type - kardinality (1:1, 1:N, N:N),
    b. onDelete – zabezpečenie RI pri odstránení (cascade, restrict, no action, set null),
    c. onUpdate – zabezpečenie RI pri odstránení (cascade, restrict, no action, set null),
5. So zdrojom a cieľom prepojenia je možné pracovať v karte (Obr. 4 - C) *Pointers.*

Obr. 4: Modelovanie prepojenia entít

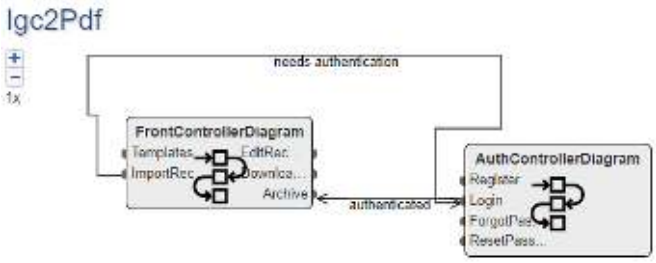
6. Modelovanie modulov

Jednotlivé moduly predstavujú väčšie celky aplikácie. Moduly v DSML sú interpretované artefaktmi typu *ControllerDiagram*. Teda každý modul je v podstate diagramom kontrolérov, ktorý môže obsahovať určité stavy.

Postup pri vytváraní diagramov modulov je nasledovný:

1. Zo zoznamu dostupných artefaktov vytvoriť inštanciu *ControllerDiagram.*
2. Nastaviť meno v sekcii *Property editor.*
3. Opakovať postup pre všetky moduly.

Ako je vidieť na Obr. 5, jednotlivé moduly sú zvyčajne medzi sebou prepojené, pričom ku prechodu z jedného modulu do druhého dochádza za určitých podmienok. Tieto podmienky môžu byť definované pri samotnom prepojení pre zlepšenie informačnej hodnoty diagramu.



Obr. 5: Modelovanie systémových modulov

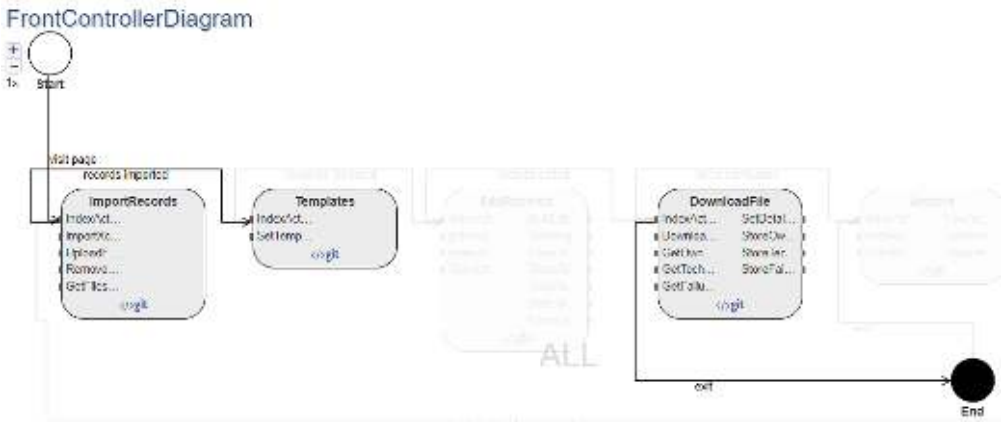
Prepojenia modulov je možné vytvoriť až v momente, kedy tieto moduly (*ControllerDiagram*) majú definované už konkrétne stavy (*Controller*), čo je obsahom kapitoly Modelovanie stavov.

Postup pri vytváraní prepojení modulov:

1. Prejdením myšou ponad modul sa zvýraznia nad portmi kontrolérov modulu oblasti.
2. Stlačením portu a potiahnutím nad port iného modulu sa vytvorí orientovaná hrana prepojenia.
3. Je vhodné nastaviť v *Property editor* aj podmienku, pri ktorej sa prechod iniciuje.
4. Postup opakovať pre všetky prepojenia v diagrame.

7. Modelovanie stavov

Stavy sa modelujú prostredníctvom kontrolérov *Controller* vo vnútri určitého modulu *ControllerDiagram.* Príklad takéhoto stavového diagramu so zvýraznenými podstatnými komponentmi je zobrazený na Obr. 6.



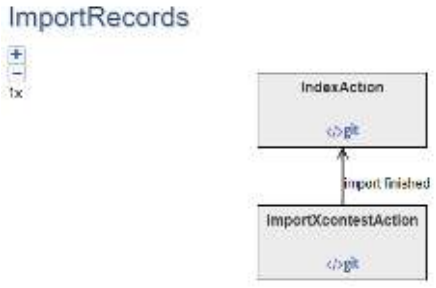
Obr. 6: Modelovanie stavov

Postup pri modelovaní stavov:

1. Zo zoznamu dostupných artefaktov vytvoríme inštanciu typu *Controller.*
2. Nastavíme atribúty v *Property editor:*
    a. name – názov kontroléra,
    b. src – nastavenie menného priestoru a názvu kontroléra.
3. Postup opakujeme pre všetky stavy.

Jednotlivé stavy a teda kontroléry zvyčajne majú definovaný jeden alebo viac pod stavov, ktoré označujeme ako akcie. Na Obr. 7 vidíme tieto akcie. Postup pri vytváraní akcií je nasledovný:

1. Zo zoznamu dostupných artefaktov vytvoríme inštanciu typu *Action.*
2. Nastavíme názov v *Property editor.*
3. Postup opakujeme pre všetky akcie.



Obr. 7: Modelovanie akcií kontroléra

Jednotlivé kontroléry majú medzi sebou zvyčajne definované prechody - viď. Obr. 6. Postup pri vytváraní týchto prechodov je nasledovný:

1. Prejdením myšou ponad kontrolér sa zvýraznia nad portmi akcií kontroléra oblasti.
2. Stlačením portu a potiahnutím k portu inej akcie kontroléra sa vytvorí orientovaná hrana.
3. Je vhodné nastaviť v *Property editor* aj podmienku, pri ktorej sa prechod iniciuje.
4. Postup opakovať pre všetky prechody v diagrame.

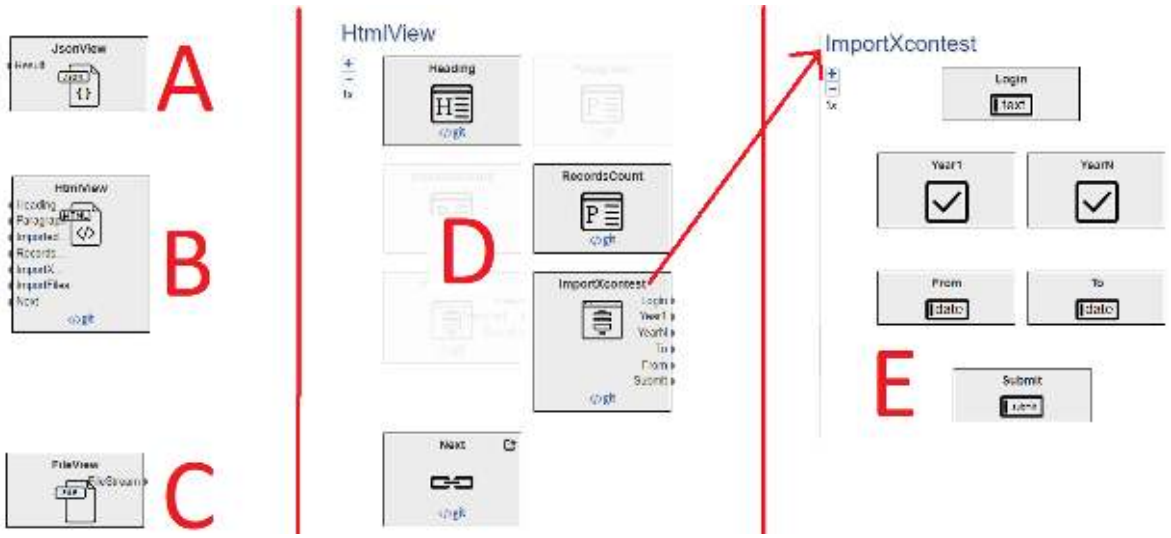
Tento postup je principiálne úplne rovnaký aj pri vytváraní prechodov medzi jednotlivými pod stavmi, teda akciami.

Pri navrhovaní kontrolérov odporúčame ich navrhovať takým spôsobom, aby jeden kontrolér obsahoval iba jednu hlavnú akciu, ktorá je určená pre vykresľovanie hlavného pohľadu pre GUI. Tieto hlavné akcie je dobré označovať napríklad ako akcie *Index. V*ýskyt viacerých takýchto akcií vykresľujúcich GUI v rámci kontroléra môže naznačovať možnú chybu alebo priestor na vylepšenia v návrhu stavov. V takom prípade odporúčame zvážiť tento kontrolér rozdeliť na viacero špecifickejších kontrolérov. V podstate ide o *single-responsibility principle* [2] z návrhového princípu SOLID pre vytváranie tried v paradigme OOP. Teda vedie to k vzniku nových stavov aplikácie.

8. Modelovanie GUI a pohľadov

Máme viacero typov pohľadov, ktoré majú svoj špecifický význam:

- Modelovanie GUI – modelujeme HTML šablónu, z ktorej sa generuje šablóna, ktorú vykresľuje webový prehliadač. Modelujeme ju pomocou *HtmlView* diagramu (Obr. 8 - B).
- Modelovanie JSON reťazca – používa sa na prenos informácií v textovej podobe. Využíva ho prehliadač prostredníctvom jazyka JavaScript na prezentovanie výsledku určitej asynchrónnej operácie, ktorá sa vykonáva po vykreslení GUI. Modelujeme ho pomocou *JsonView* diagramu (Obr. 8 - A). Konkrétne sa napríklad používa pri notifikácii používateľa o výsledku nahrávania súboru bez nutnosti znovu načítania stránky.
- Modelovanie súboru – v prípade, že výstupom danej akcie je nejaký súbor, na jeho modelovú reprezentáciu použijeme diagram *FileView* (Obr. 8 - C).



Obr. 8: Modelovanie pohľadu

Ďalej budeme pokračovať najkomplexnejším pohľadom viď. D - Obr. 8, ktorým je šablóna pre GUI. Tá obsahuje množstvo HTML elementov, ktoré sú dostupné v kontexte *HtmlView* diagramu. Postup pri modelovaní šablóny je nasledovný:

1. V akcii je potrebné vytvoriť inštanciu diagramu *HtmlView* a otvoriť ho.
2. Zo zoznamu dostupných artefaktov vytvoríme inštanciu pre požadovaný typ elementu.
3. Nastavíme jeho atribúty v *Property editore.* Každý HTML element má špecifické atribúty, no nebudeme ich všetky uvádzať, pretože vývojár so základnými znalosťami webov bude vedieť čo znamenajú a ako ich nastaviť.
4. Postup opakujeme pre všetky želané atribúty od bodu 2.

Formuláre tvoria zložitejšiu štruktúru, preto im bol vytvorený osobitný diagram *FormularDiagram*, v ktorom sa definujú prvky tohto formulára. Princíp postupu modelovania je totožný s predchádzajúcim postupom.

Ďalším špecifikom tohto diagramu sú hypertextové odkazy, ktoré môžu smerovať na určitú konkrétnu akciu tohto kontroléra. Používateľ po kliknutí na tento odkaz je na túto akciu presmerovaný. Model odkazu obsahuje aj v pravom hornom rohu šípku viď. Obr. 9. Po dvoj kliknutí na túto šípku sa prepne kontext do cieľovej akcie odkazu.

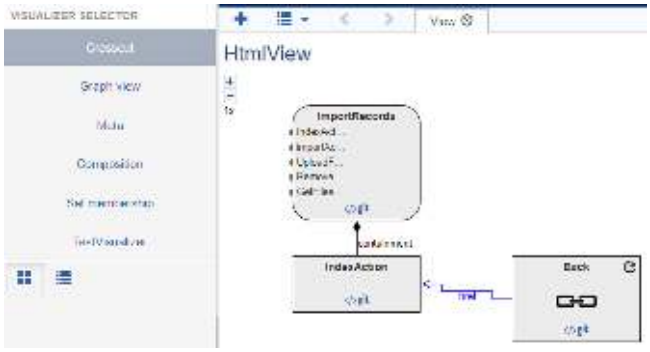
Definovanie cieľového pointra odkazu:

1. Na pravej strane editora v sekcii *Object browser* viď. Obr. 9 v preferovanom type stromového zobrazenia je potrebné nájsť požadovanú cieľovú akciu.
2. Túto akciu je potrebné pretiahnuť smerom nad uzol požadovaného hypertextového odkazu.
3. Opakovať pre všetky hypertextové odkazy

Obr. 9 Vytváranie hypertextových odkazov

Nástroj poskytuje aj možnosť prehľadného zobrazenia prepojení všetkých hypertextových odkazov na ich cieľové akcie viď. Obr. 10 prepnutím sa do zobrazenia *Crosscut*.

Takýto pohľad je ale potrebné vytvoriť a nadefinovať požadované artefakty tým, že ich pretiahneme zo stromu *Object browser* do vizualizéra.

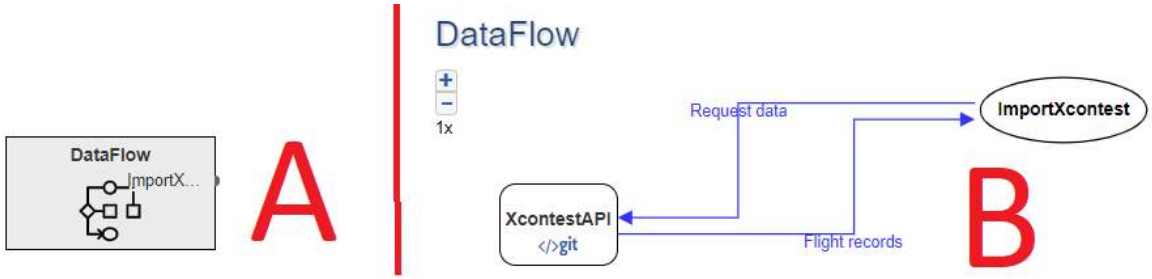


Obr. 10: Zobrazenie hypertextových prepojení šablóny

9. Modelovanie dátového toku

Dátový tok definujeme samostatne pre každú akciu kontroléra. Postup pri modelovaní dátového toku je nasledovný:

1. Je potrebné v akcii kontroléra vytvoriť inštanciu diagramu *DataFlow* viď. „A“ - Obr. 11
2. Zo zoznamu dostupných artefaktov vytvoríme inštanciu pre požadovaný typ uzla avšak vynecháme *DataStore* modelujúce databázové entity a taktiež vynecháme aj uzly, ktoré patria pohľadu. K týmto artefaktom sa vrátime neskôr.
3. Zadefinujeme jeho názov a ďalšie atribúty v *Property editor.*
4. Opakujeme pre všetky požadované elementy.

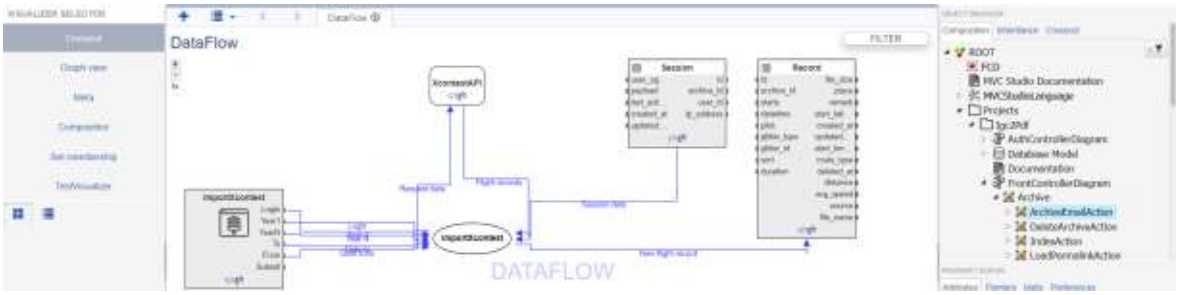


Obr. 11: Modelovanie toku dát – čiastočný model

Kvôli nevýhode WebGME, ktorá neumožňuje znovu použitie určitého uzla naprieč modelom sme nútení tento problém obísť pomocou vizualizéra *Crosscut.* Pri pokuse o opätovné použitie uzla sa vytvorí vo WebGME nová inštancia dediaca od tohto uzla, pričom tento efekt je neželaný a spôsoboval by mnoho komplikácií hlavne pri automatizácii.

Náš model dátového toku teda dokončíme nasledujúcim postupom:

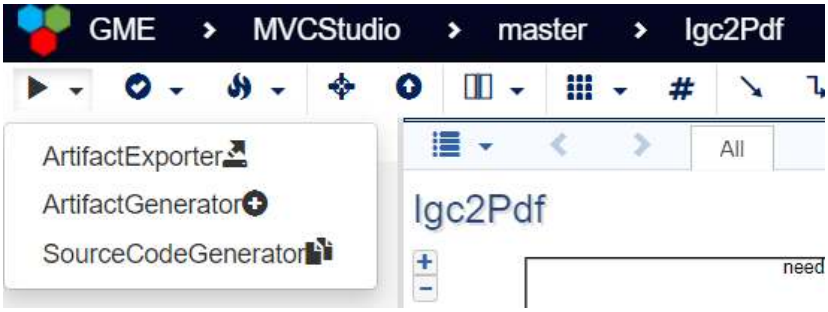
1. Prepneme sa do vizualizéra *Crosscut* viď. Obr. 12.
2. V sekcii *Object browser* vyberieme všetky uzly, ktoré má obsahovať.
3. Poprepájame tieto uzly.
4. Prepojeniam definujeme charakter prenášanej informácie.



Obr. 12: Modelovanie toku dát – kompletný model

10. Rozšírenia

Nástroj MVCStudio prichádza aj s určitými implementovanými rozšíreniami, ktoré je možné iniciovať v ľavej hornej časti obrazovky viď. Obr. 13.



Obr. 13: Spustenie rozšírení

Pri niektorých modeloch sú na spodnej časti zobrazené tlačidlá *git a </>* viď. Obr. 14. Po kliknutí na tlačidlo bude otvorená reprezentácia modelu v zdrojovom kóde buď v projekte na gitlabe alebo priamo v hosťujúcom počítači vo vývojovom prostredí. Druhú funkciu je pred použitím potrebné nakonfigurovať, čo je obsahom systémovej príručky.

Obr. 14: Prechod od modelu k zdrojovému kódu

Zdroje

[1]. FRANTIŠEK FERENČÍK: Metamodelovanie, modelovanie, verifikácia a validácia požiadaviek. Diplomová práca, Technická univerzita v Košiciach, 2020.

[2]. MADASU, VAMSI & VENNA, TRINADH & ELTAEIB, TARIK: SOLID Principles in Software Architecture and Introduction to RESM Concept in OOP, 2015, Journal of Engineering Science and Technology. 2. 3159-40., [cit. 2020-04-26], Dostupné online: < https://www.researchgate.net/publication/273451390_SOLID_Principles_in_Software_Architecture_and_Introduction_to_RESM_Concept_in_OOP>.

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DEMONŠTRÁCIA DSML NA SYSTÉME IGC2PDF

Príloha D

2020 **František Ferenčík**

Obsah

Zoznam obrázkov

Zoznam Tabuliek

1. Súpis obsahu dodávky

CD priložené k diplomovej práci obsahuje:

- Zdrojové kódy aplikácie,
- systémovú príručku,
- metodickú a používateľskú príručku,
- demonštráciu DSML na systéme igc2pdf.

2. Systém igc2pdf

Obsahom tejto prílohy je zhrnutie informácií ohľadom aplikácie igc2pdf, ktorá je v tejto diplomovej práci použitá na účely demonštrácie vytvoreného CASE nástroja MVCStudio implementujúceho DSML pre vývoj webových aplikácií založených na architektúre MVC. Požiadavky sú čerpané najmä z bakalárskej práce [1] , v ktorej bola táto aplikácia vyvinutá.

2.1. Popis systému

Systém igc2pdf je webová aplikácia zameraná na uľahčenie administratívnych úkonov pilotom LŠZ (letové športové zariadenie) pri dokumentovaní svojich letov pomocou záznamníkov. Letecká amatérska asociácia Slovenskej republiky LAA SR si podľa legislatívy vyžaduje viesť takúto dokumentáciu. Prostredníctvom tohto systému si pilot môže importovať letové záznamy zo súborov s formátom IGC alebo prostredníctvom hromadného importu z portálu xcontest.org. Letové záznamy je možné upravovať. Letová dokumentácia je vygenerovaná s požadovanou zvolenou šablónou a prístupná na stiahnutie vo formáte PDF. Podporované sú aj viaceré možnosti archivácie práce v tomto systéme.

Táto aplikácia bola navrhnutá a zrealizovaná v roku 2018 v rámci bakalárskej práce [1]. Video ukážka aplikácie sa nachádza na odkaze [2]. V popise videa sa nachádza aj aktuálny odkaz na aplikáciu nasadenú v produkčnom prostredí.

2.2. Technické požiadavky a závislosti

Technické požiadavky na hosťujúce serverové prostredie sú:

- PHP Composer,
- Platforma LAMP,
    - Linux operačný systém,
    - Webový server Apache,
    - MySQL databázový server,
    - Programovací jazyk PHP,
        - Verzia PHP >= 7.0.0,
        - OpenSSL PHP extension,
        - PDO PHP extension,
        - Mbstring PHP extension,
        - Tokenizer PHP extension,
        - XML PHP extension,
    - PHP Composer.

3. Zoznam požiadaviek systému

Táto kapitola sumarizuje funkčné a nefunkčne požiadavky na systém igc2pdf [1].

3.1. Funkčné požiadavky

Základné funkčné požiadavky kladené na klientsku aplikáciu sú zhrnuté v Tab. 1.

Tab. 1: Funkčné požiadavky systému igc2pdf

Číslo	**Požiadavka**
FR1	Možnosť importovať letové záznamy
FR1.1	Importovanie zo súborov vo formáte IGC
FR1.1.1	Možnosť vymazania nechceného súboru
FR1.2	Importovanie letových záznamov z portálu xcontest.org
FR1.2.1	Import pilotových záznamov na základe prihlasovacieho mena portálu xcontest.org
FR1.2.2	Pri importe si môže zvoliť požadované súťažné roky
FR1.3	Zobrazovať priebežný stav importovania súborov
FR2	Výber medzi dostupnými šablónami textových dokumentov
FR2.1	Záznamník pilota LŠZ
FR2.2	Záznamník LŠZ
FR2.2.1	Výber konkrétneho LŠZ, pre ktorý sa bude záznamník generovať
FR3	Zobrazenie zoznamu letových záznamov
FR4	Editácia letových záznamov
FR4.1	Vytvorenie nového letového záznamu
FR4.2	Úprava obsahu už existujúceho letového záznamu
FR4.3	Vymazanie nechceného letového záznamu
FR4.4	Zobrazenie upozornení na problémy letových záznamov (nevyplnené pole)
FR4.5	Identifikácia miesta štartu podľa súradníc štartu
FR4.6	Vloženie miesta letu do poznámky
FR4.7	Vloženie štátu letu do poznámky
FR4.8	Vloženie času vzletu a pristátia do poznámky
FR4.9	Vloženie typu a dĺžky trate do poznámky
FR4.10	Hromadná editácia záznamov pre vybraný stĺpec
FR5	Generovanie letového záznamníka na základe zvolenej šablóny dokumentu
FR5.1	Stiahnutie záznamníka vo formáte PDF
FR6	Možnosť prispôsobenia záznamníka

FR6.1	Možnosť spájať riadky na základe dátumu
FR6.2	Nastavenie počiatočného čísla stránkovania dokumentu
FR6.3	Nastavenie prenosu z predošlej strany – štarty, hodiny a minúty
FR6.4	Nastavenia obsahu titulných strán jednotlivých šablón dokumentov
FR7	Zaznamenávanie informácií o majiteľovi LŠZ
FR7.1	Umožniť pridávanie, editáciu, mazanie a zobrazovanie v príslušných šablónach
FR8	Zaznamenávanie technických informácií LŠZ
FR8.1	Umožniť pridávanie, editáciu, mazanie a zobrazovanie v príslušných šablónach
FR9	Zaznamenávanie porúch LŠZ
FR9.1	Umožniť pridávanie, editáciu, mazanie a zobrazovanie v príslušných šablónach
FR10	Archivácia práce
FR10.1	Archivácia na e-mail s PDF prílohou záznamníkov
FR10.2	Archivácia pomocou trvalého odkazu
FR10.2.1	Možnosť načítať archív pomocou trvalého odkazu a pokračovať v práci
FR10.3	Archivácia do používateľského konta
FR10.3.1	Zobrazenie zoznamu archívov
FR10.3.2	Možnosť načítať archív a pokračovať v práci
FR10.3.3	Možnosť odstránenia nepotrebného archívu
FR11	Registrácia používateľa
FR12	Prihlásenie používateľa
FR13	Možnosť požiadať o zmenu zabudnutého hesla
FR13.1	Možnosť vykonať zmenu zabudnutého hesla

3.2. Nefunkčné požiadavky

Zoznam nefunkčných požiadaviek na systém igc2pdf je zhrnutý v Tab. 2.

Tab. 2: Nefunkčné požiadavky na systém igc2pdf

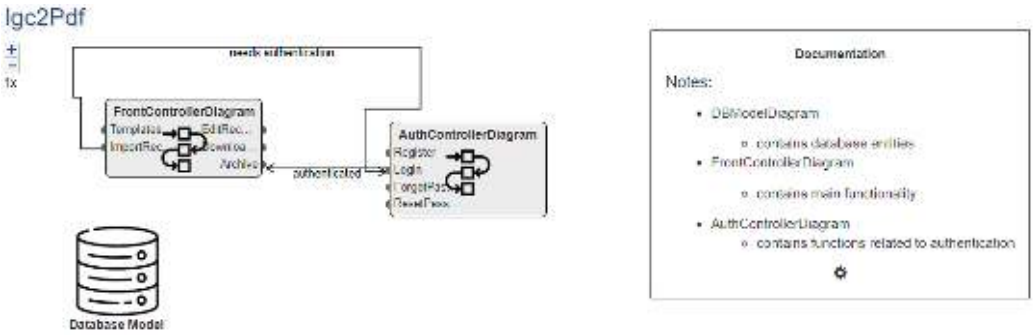
Číslo	**Požiadavka**
NR1	Návrh musí byť s dôrazom na UX (user experience)
NR2	Používateľské rozhranie musí byť intuitívne
NR3	Dostupnosť použitých technológií
NR3.1	Uprednostniť technológie open-source
NR4	Aplikácia musí byť vyvíjaná s dôrazom na budúce zmenové a doplňujúce úpravy

4. Kompletný zoznam DSML modelov systému

Obsahom tejto kapitoly je súhrn modelov vytvorených pomocou DSML navrhnutého v rámci tejto diplomovej práce. Celé modelovanie dát je popísané spolu v jednom celku ale modelovanie riadenia stavov, grafického používateľského rozhrania a dátového toku bude čiastkovo popisované vzhľadom na jednotlivé funkčné celky.

4.1. Balík projektu

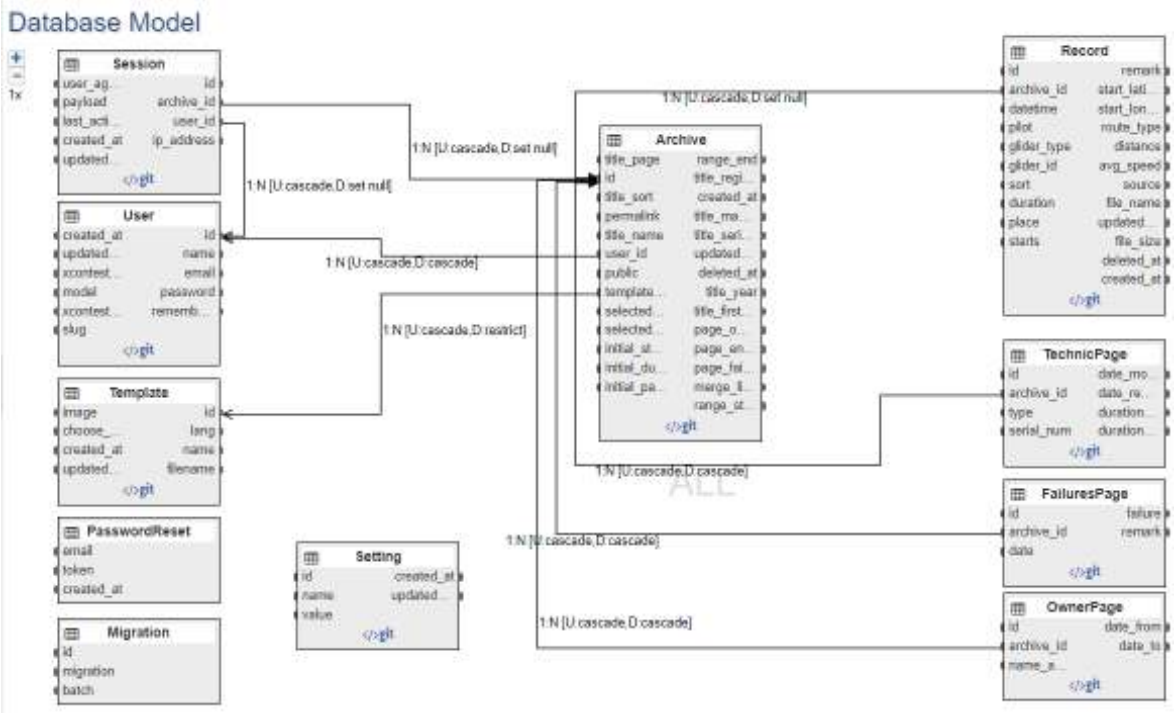
Na Obr. 1 je znázornený balík projektu obsahujúci model dátovej vrstvy, diagramy modulov, kontrolérov a poznámky v rámci dokumentácie. Jednotlivé modely samozrejme obsahujú ďalšie vnorené modely pričom systém dekomponujú systémom zhora nadol teda od komplexných celkov až po ich detaily.



Obr. 1: Hlavný balík projektu Igc2Pdf.

4.2. Dátová vrstva - ModelDiagram

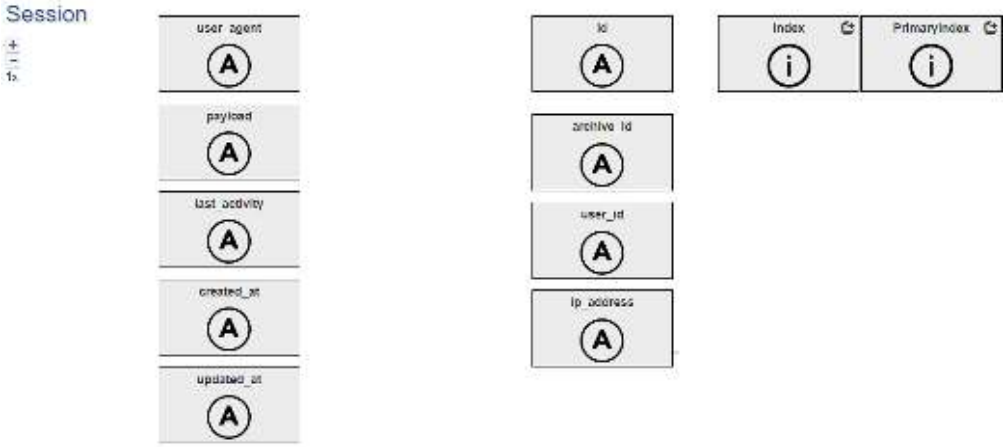
ModelDiagram zobrazený na Obr. 2 zachytáva dátovú vrstvu, ktorú systém využíva na manipulovanie a perzistovanie informácií. Obsahuje jednotlivé databázové entity s ich atribútmi a zároveň zobrazuje prepojenia medzi nimi spolu s definovaním pravidiel zachovania referenčnej integrity pre akcie aktualizácie a vymazania kľúčového záznamu.



Obr. 2: ModelDiagram databázovej vrstvy

4.2.1.Session entita

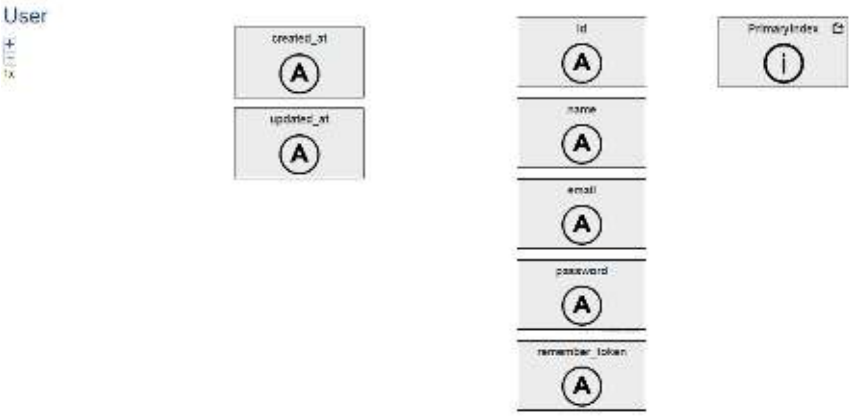
V tejto entite sú perzistované údaje príslušnej relácie. Inač povedané obsahuje dáta reprezentujúce stav používateľa pri používaní aplikácie. Na Obr. 3 sú namodelované atribúty a indexy entity *Session.*



Obr. 3: Session entita

4.2.2.User entita

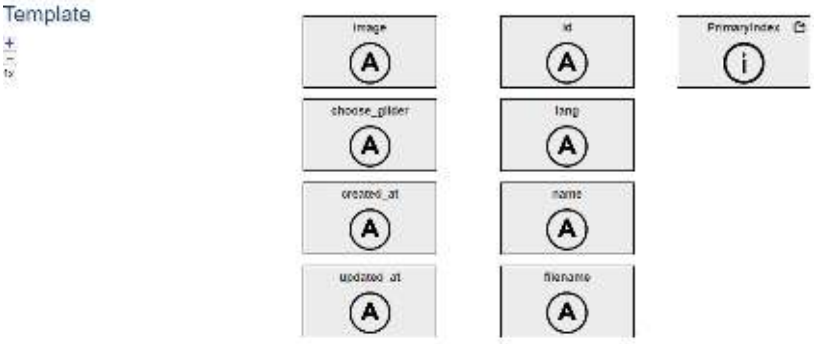
Táto databázová entita perzistuje informácie o zaregistrovanom používateľovi. Model je zobrazený na Obr. 4.



Obr. 4: User entita

4.2.3.Template entita

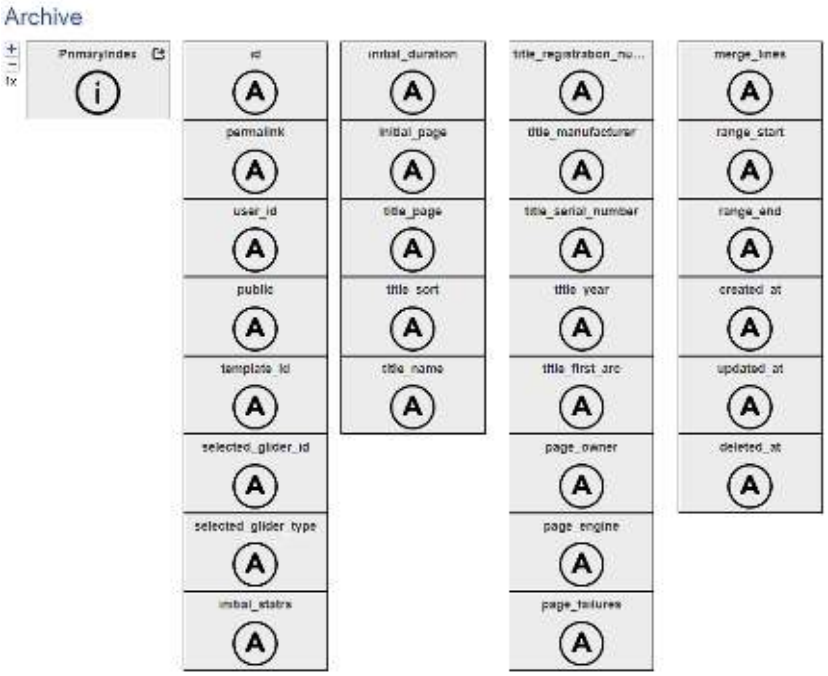
Šablóny letovej dokumentácie sú reprezentované modelom *Template* zobrazenom na Obr. 5.



Obr. 5: Template entita

4.2.4.Archive entita

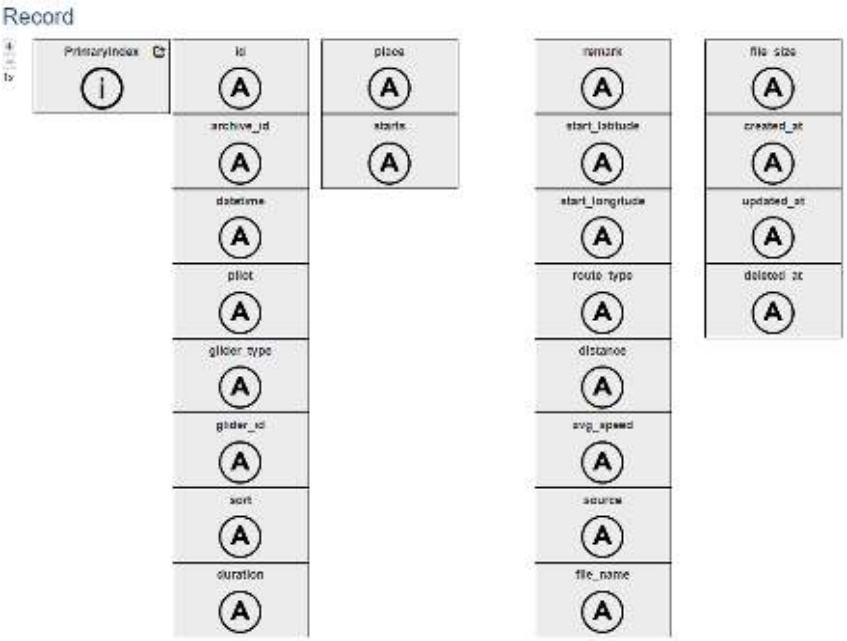
Za archív sa považuje skupina dát spolu súvisiaca. Tento archív namodelovaný na Obr. 6 zlučuje všetky konfigurácie v rámci príslušnej relácie.



Obr. 6: Archive entita

4.2.5. Record entita

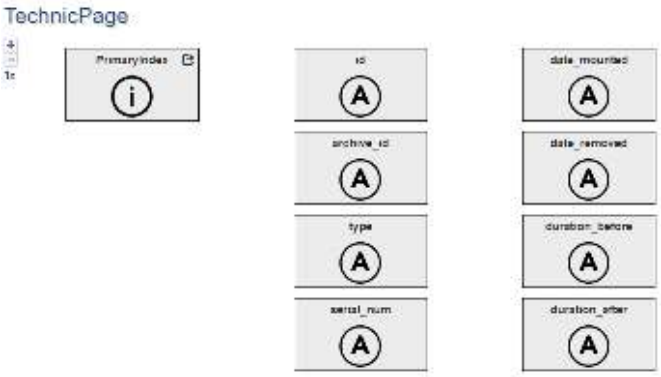
Samotné letové záznamy sú perzistované v entite Record, ktorá je namodelovaná na Obr. 7.



Obr. 7: Record entita

4.2.6. TechnicPage entita

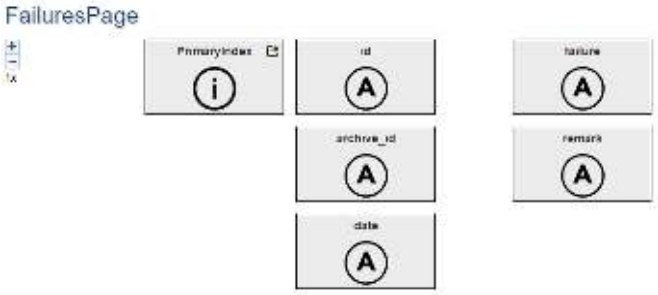
Záznamy o technických kontrolách sa perzistujú v tabuľke *TechnicPage* namodelovanej na Obr. 8.



Obr. 8: *TechnicPage* entita

4.2.7. FailuresPage entita

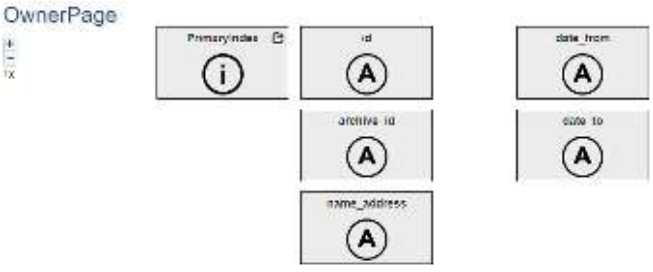
Dokumentácia porúch sa ukladá v tabuľke *FailuresPage* namodelovanej na Obr. 9.



Obr. 9: *FailuresPage* entita

4.2.8. OwnerPage entita

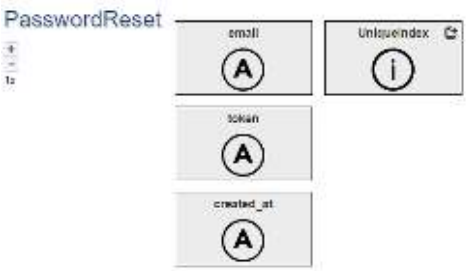
Údaje o majiteľoch letových športových zariadení sú perzistované v tabuľke *OwnerPage*, ktorej model je zobrazený na Obr. 10.



Obr. 10: *OwnerPage* entita

4.2.9. PasswordReset entita

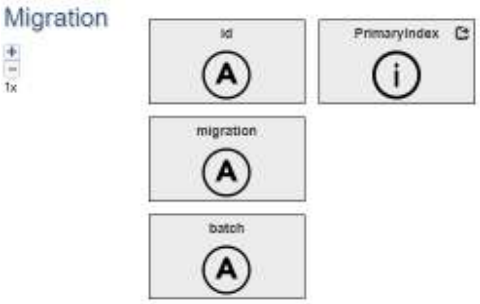
Požiadavky na zmenu hesla používateľského účtu sú perzistované v tabuľke *PasswordReset* namodelovanej na Obr. 11.



Obr. 11: *PasswordReset* entita

4.2.10. Migration entita

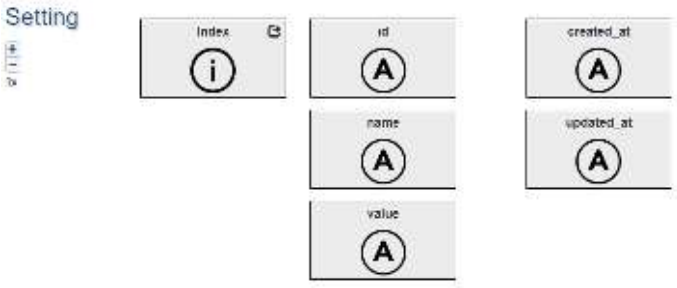
Evidencia databázových migrácií sa nachádza v tabuľke *Migration*, ktorá je namodelovaná na Obr. 12.



Obr. 12*:* Migration entita

4.2.11. Setting entita

Globálne nastavenia a iné dôležité všeobecné informácie sú perzistované v tabuľke *Setting*, ktorej model je zobrazený na Obr. 13.

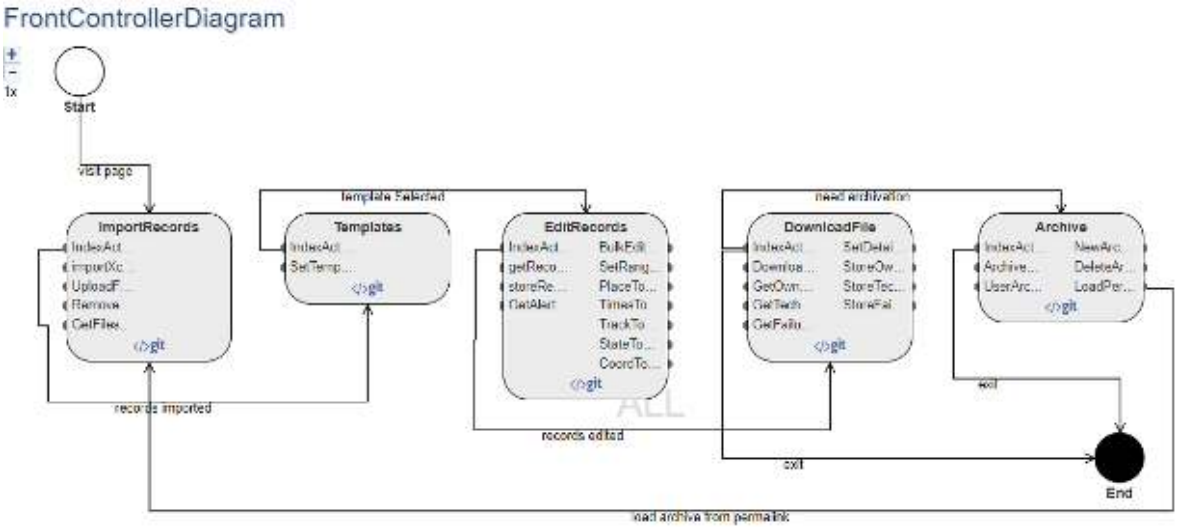


Obr. 13: Setting entita

4.3. FrontControllerDiagram – hlavná časť systému

Hlavnú časť systému, označenú na Obr. 1 ako *FrontControllerDiagram* reprezentujú všetky funkcie a obrazovky prístupné každému používateľovi bez obmedzenia. Jeho model sa nachádza na Obr. 14. Sú tu namodelované hlavné funkčné celky zlúčené do jednotlivých kontrolérov. Prechod medzi jednotlivými kontrolérmi je namodelovaný prepojením s popisom, kedy dochádza ku prechodu a teda ku zmene stavu aplikácie. Je tu taktiež definovaný počiatočný a koncový stav.

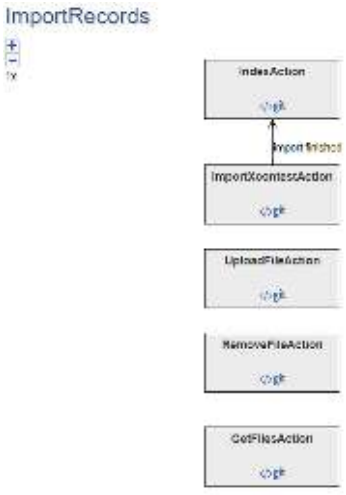
Po návšteve stránky sa používateľ dostane na obrazovku, kde môže importovať svoje letové záznamy. Potom nasleduje obrazovka s výberom šablóny záznamníka. Po výbere sa dostane v ďalšom kroku do sekcie pre úpravu letových záznamov. Ďalším krokom je obrazovka s možnosťou nastavenia a stiahnutia dokumentácie v PDF. Z tejto obrazovky môže prácu buď ukončiť alebo pokračovať k funkciám archivácie na ďalšej obrazovke. Po ukončení archivácie prácu môže ukončiť. Tento kontrolér sa taktiež stará o načítanie archívu z trvalého odkazu, pričom používateľ je presmerovaný na úvodnú obrazovku.



Obr. 14: *FrontControllerDiagram* – stavový model hlavných funkcií systému

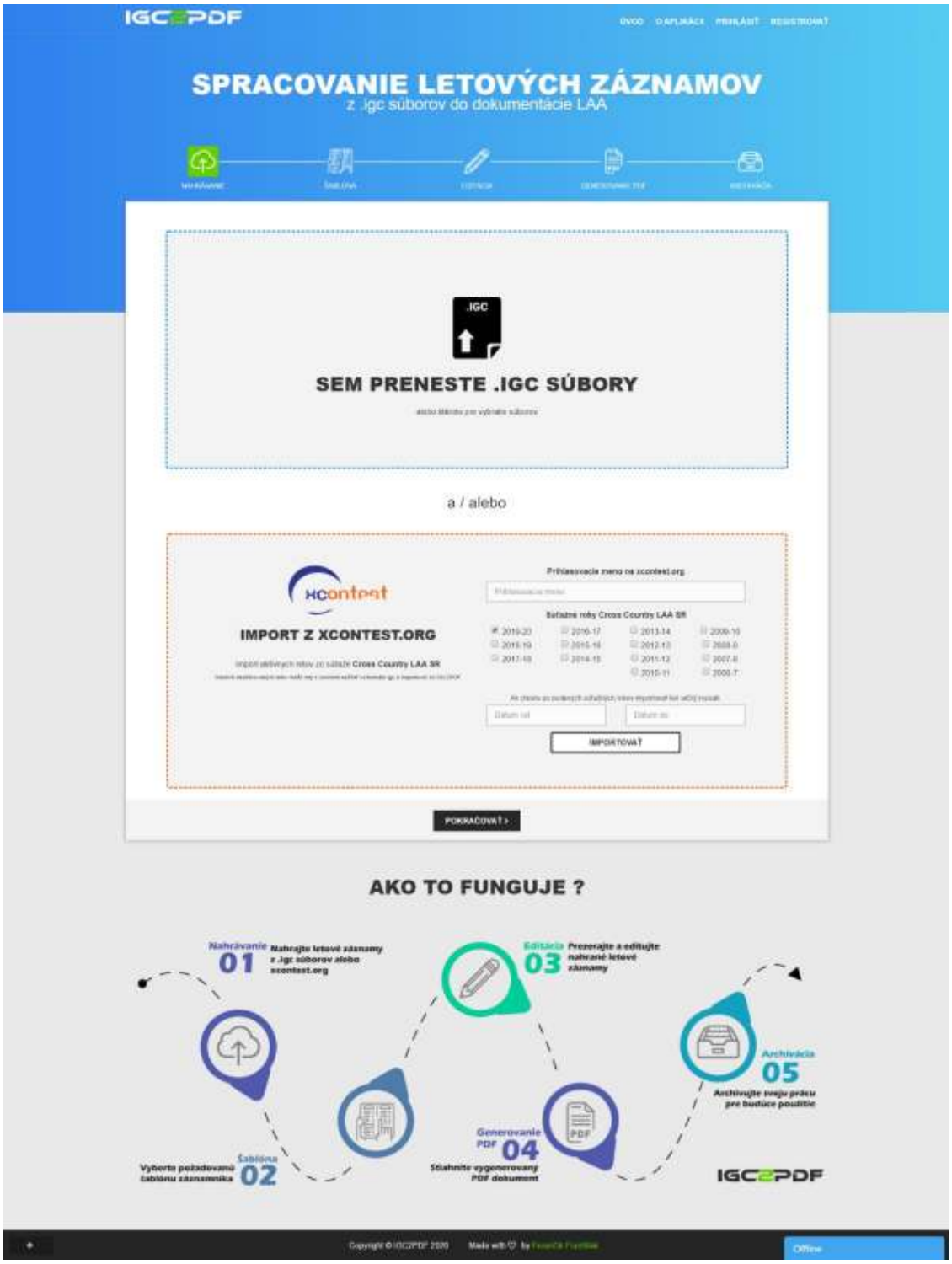
4.3.1. ImportRecords kontrolér – importovanie letových záznamov

Tento kontrolér združuje všetky systémové funkcie zabezpečujúce import letových záznamov či už z IGC súborov alebo z externého servera xcontest.org. Táto obrazovka je zachytená na Obr. 16.



Obr. 15: Akcie kontroléra *ImportRecords*.

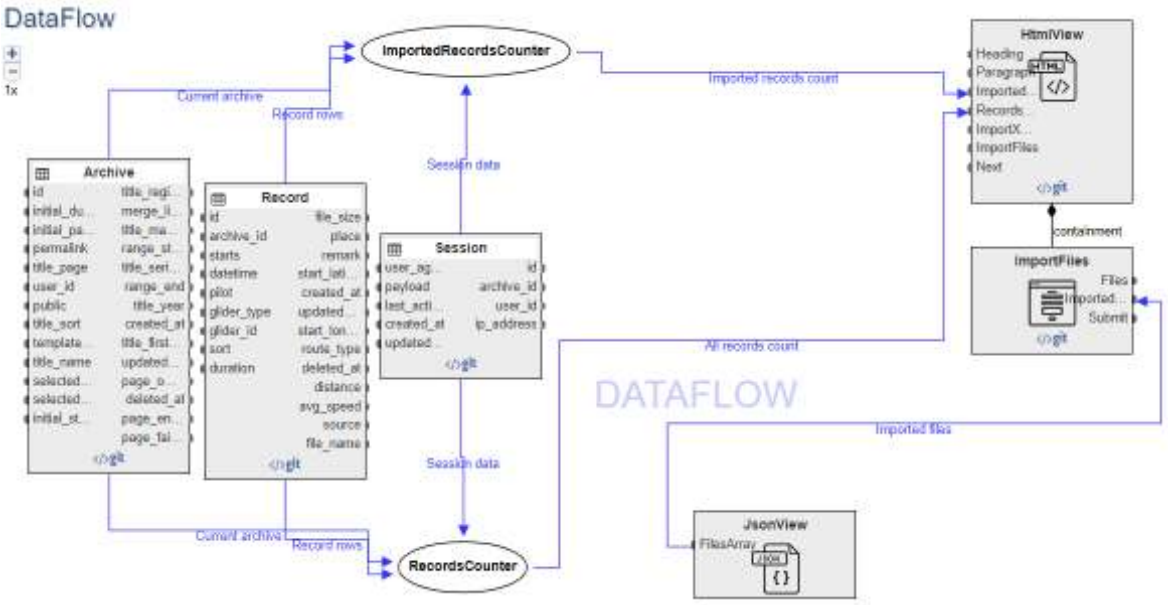
Na Obr. 15 sú znázornené akcie, teda čiastkové podstavy zabezpečujúce rôzne funkcie tejto obrazovky. Akcie môžu tiež obsahovať prepojenia, ktorých popis vyjadruje udalosť, po ktorej je stav aplikácie zmenený v smere šípky.



Obr. 16: 1. stav - obrazovka pre nahrávanie letových záznamov

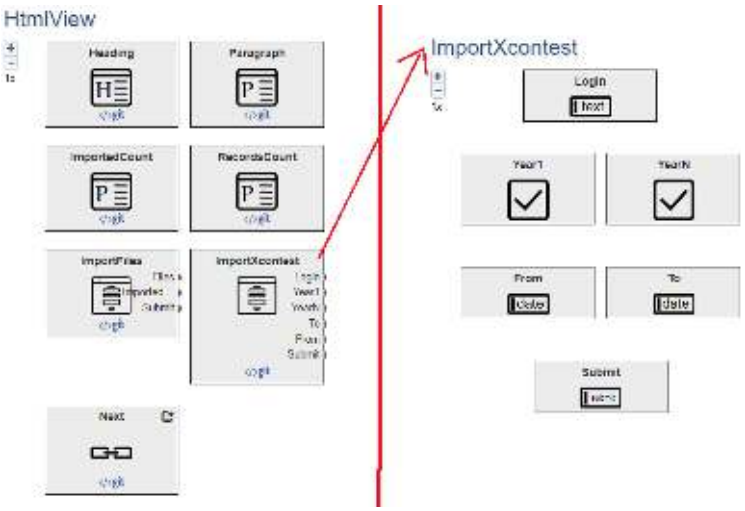
4.3.1.1. IndexAction – hlavná obrazovka importu letových záznamov

Táto akcia má na starosti vykreslenie hlavnej obrazovky zobrazenej na Obr. 16. S tým súvisí namodelovanie patričných šablón pomocou *ViewDiagram*, pričom šablóna potrebuje určité dáta, ktorých pôvod je namodelovaný pomocou diagramu dátového toku *DataFlowDiagram*. Dátový tok je namodelovaný na Obr. 17.



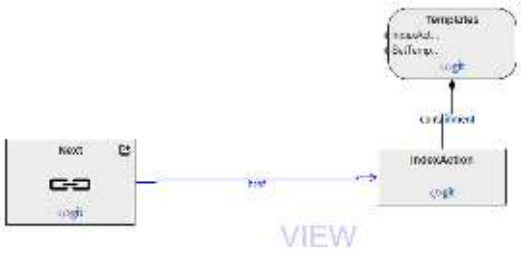
Obr. 17: *DataFlowDiagram* akcie index kontroléra *ImportRecords*

Obr. 18 zachytáva model šablóny pre modelovanie pohľadu. Táto obrazovka teda obsahuje nejaký nadpis, podnadpis, počítadla záznamov, tlačidlo pre pokračovanie na ďalší krok, formulár pre nahrávanie súborov a formulár pre import z portálu xcontest.org. Model tohto formulár nachádzajúci sa na pravej strane obrázka obsahuje ďalšie HTML elementy. Takto sú modelované všetky formuláre.



Obr. 18: *ViewDiagram* šablóny pre vykreslenie akcie index kontroléra *ImportRecords*

Súčasťou šablóny je hypertextový odkaz, ktorého prepojenie je namodelované na Obr. 19. Po prepnutí do vizualizéra *crosscut* sa objaví model vizualizujúci smer prepojení hypertextových odkazov. Takže tlačidlo pokračovať v tomto prípade smeruje na ďalší krok do kontroléra *Templates*.

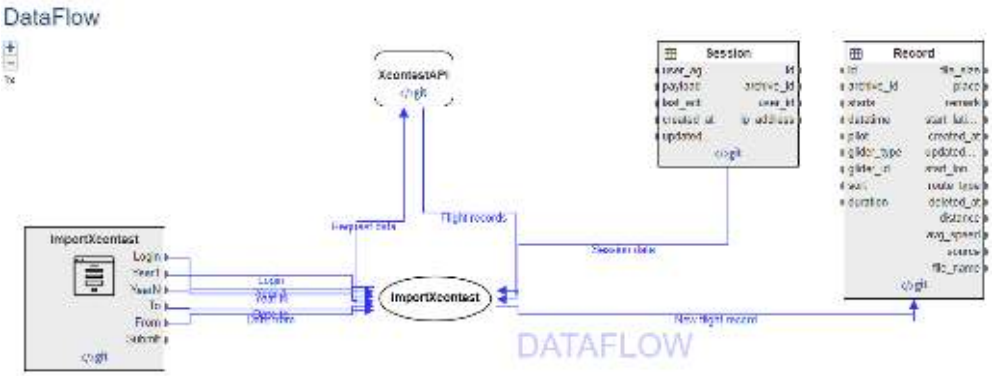


Obr. 19: Model hypertextového prepojenia v šablóne kontroléra *ImportRecords*

4.3.1.2. ImportXcontestAction – import letových záznamov z xcontestu

Táto akcia nerenderuje žiadnu formu pohľadu ale po skončení presmeruje stav na *IndexAction*. Z toho dôvodu si uvedieme len model diagramu dátového toku pri importe letových záznamov z xcontestu, ktorý je znázornený na Obr. 20.

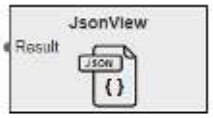
Na modeli môžeme vidieť, že proces preberie potrebné identifikácie pilota z formulára (artefakt z diagram *View*) a databázovej entity *Session* (diagram *Model*). Z týchto dát je vyskladaná požiadavka, ktorá je odoslaná na externú entitu API servera. Server vráti zoznam letových záznamov, ktorý je spracovaný vhodným spôsobom a perzistovaný do databázovej entity *Record.*



Obr. 20: *DataFlowDiagram* importu letových záznamov v akcii *ImportXcontestAction*

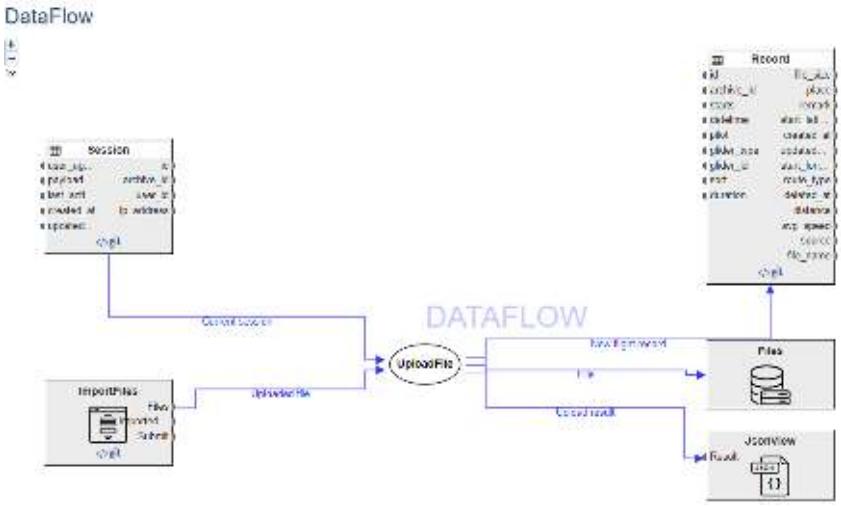
4.3.1.3. UploadFileAction – import letových záznamov zo súborov IGC

Model dátového toku nahrávania importu letových záznamov z IGC súborov je zobrazený na Obr. 22.



Obr. 21: JsonViewDiagram obsahujúci informáciu o výsledku importu súborov v *ImportRecord*

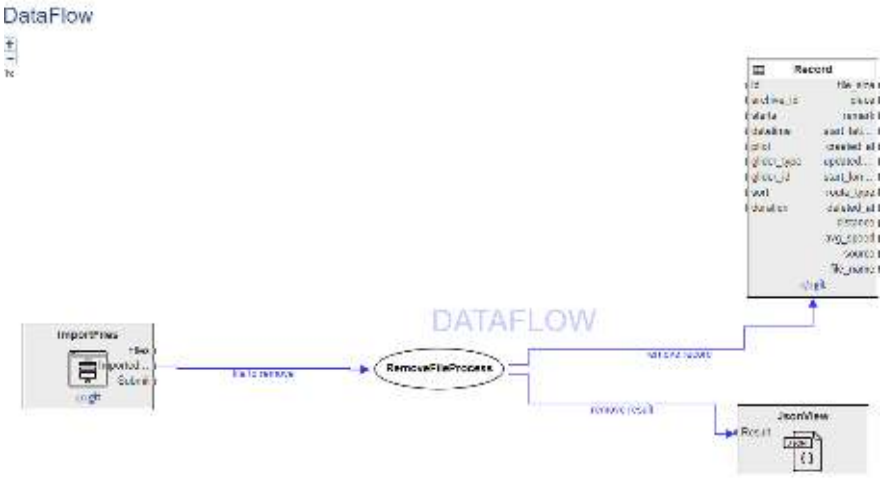
Výsledok tohto procesu je zaslaný používateľovi vo formáte JSON, pričom webový prehliadač informácie patričným spôsobom prezentuje používateľovi. Model tohto typu pohľadu je znázornený na Obr. 21. Tento typ pohľadu je viditeľný aj v *DataFlow* diagramoch, takže nebude ďalej uvádzaný samostatne.



Obr. 22: *DataFlowDiagram* akcie *UploadFileAction* kontroléra *ImportRecords*

4.3.1.4. RemoveFileAction – odstránenie nahraného IGC súboru

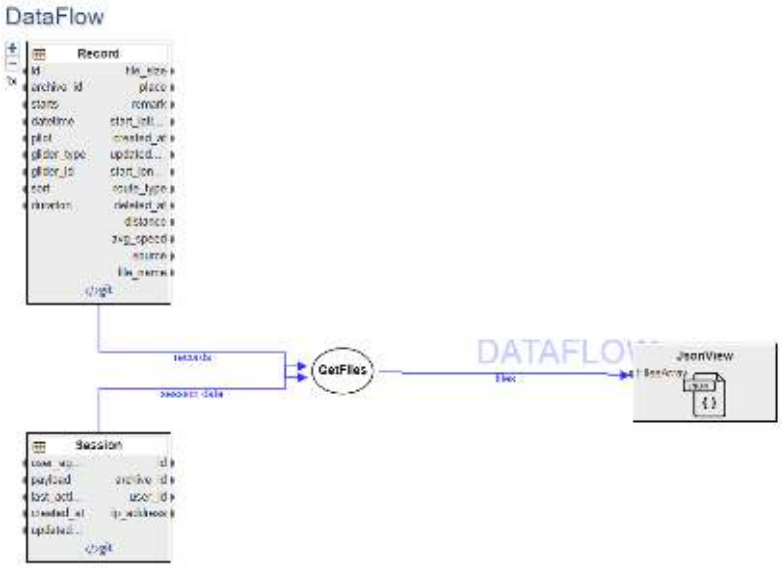
Model dátového toku pre funkciu odstránenia nahraného IGC súboru sa nachádza na Obr. 23. Výsledok tejto operácie spracovaný je vrátený pohľadom v *JsonView*.



Obr. 23: *DataFlowDiagram* odstránenia nahraného IGC súboru

4.3.1.5. GetFilesAction – získanie importovaných IGC letových záznamov

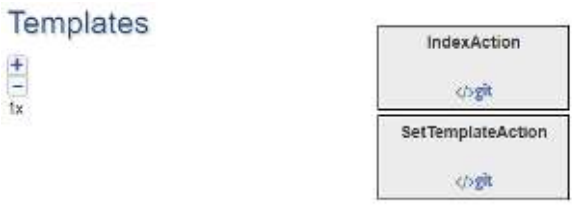
Model dátového toku získania už importovaných letových záznamov je na Obr. 24. Na obrázku je vidieť taktiež aj model pohľadu *JsonView*, pomocou ktorého sa tieto súbory prenášajú.



Obr. 24: *DataFlowDiagram* získania importovaných IGC súborov

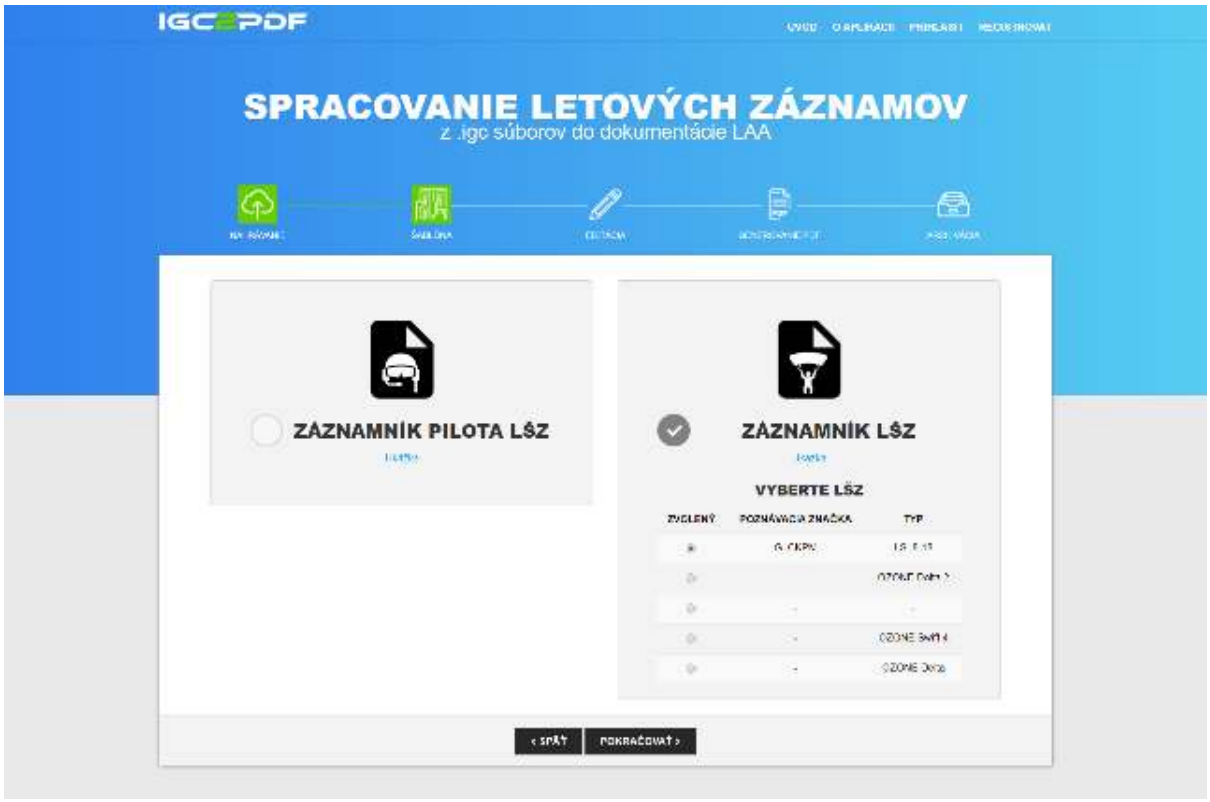
4.3.2. Templates – výber šablóny letových záznamníkov

Tento kontrolér zlučuje celú funkcionalitu ohľadom výberu šablón pre výsledné vygenerované letové dokumentácie. Tak isto umožňuje výber špecifického klzáku. Model akcií tohto kontroléra je zobrazený na Obr. 25.



Obr. 25: Akcie kontroléra *Templates*

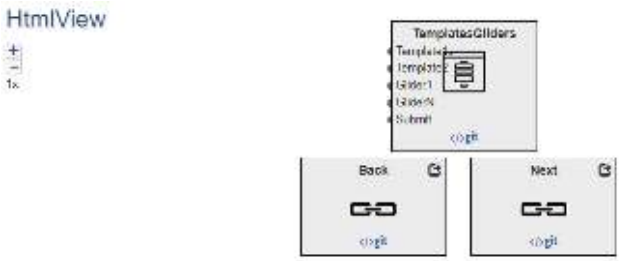
Pre lepšiu predstavu uvádzame aj snímok obrazovky tohto kontroléra na Obr. 26.



Obr. 26: 2. stav – obrazovka výber šablóny záznamníka a LŠZ

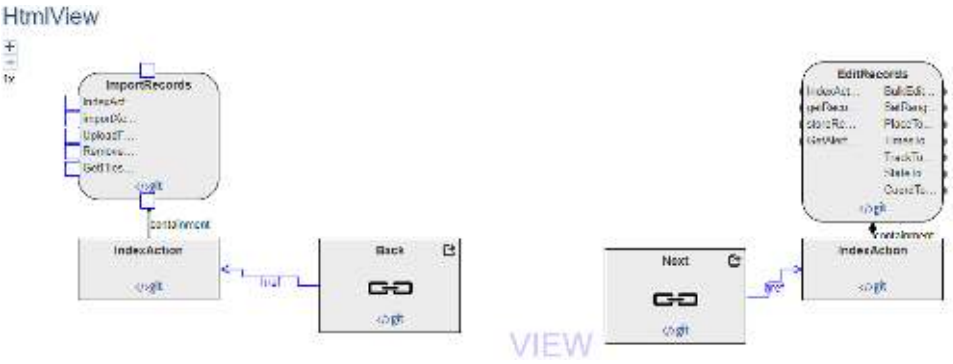
4.3.2.1. IndexAction – obrazovka pre výber záznamníka

Táto akcia je určená na vykreslenie obrazovky znázornenej na Obr. 26. Model GUI pohľadu sa nachádza na Obr. 27.



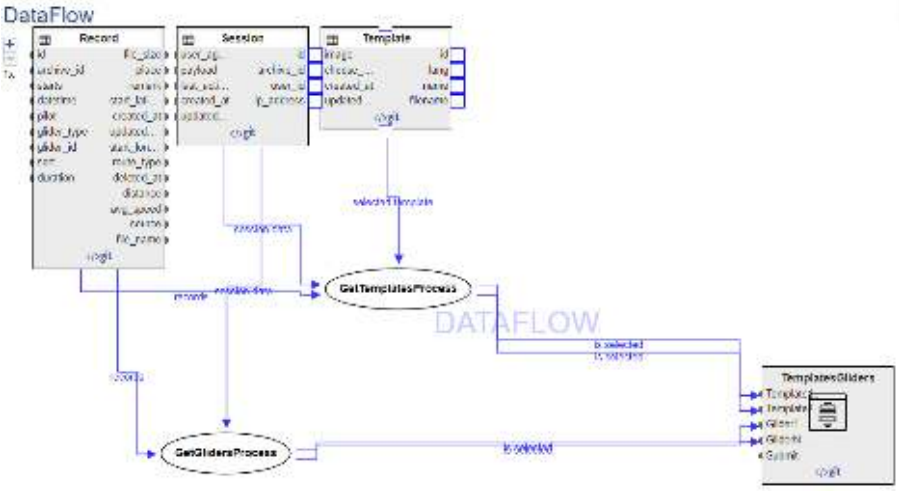
Obr. 27: Diagram pohľadu akcie index kontroléru *Templates*

Po prepnutí sa do zobrazenia *crosscut* na vidíme model reprezentujúci prepojenie hypertextových odkazov na konkrétne akcie.



Obr. 28: Hypertextové prepojenie pohľadu *IndexAction* kontroléra *Templates*

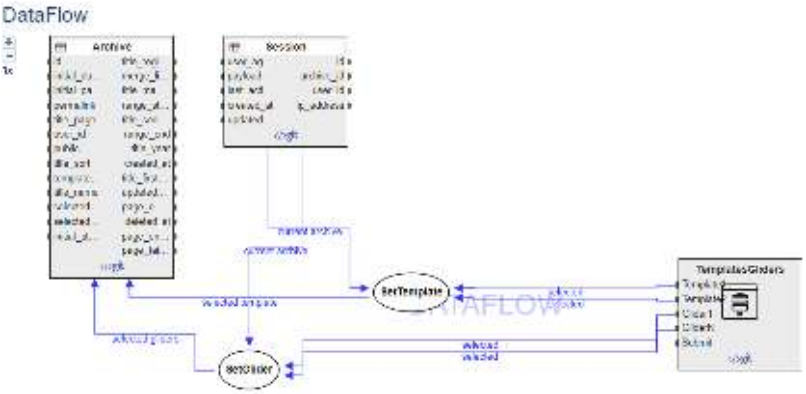
Na Obr. 29 je namodelovaný diagram dátového toku tejto akcie.



Obr. 29: Diagram dátového toku akcie *IndexAction* kontroléra *Templates*

4.3.2.2. SetTemplateAction – nastavenie príslušného záznamníka a LŠZ

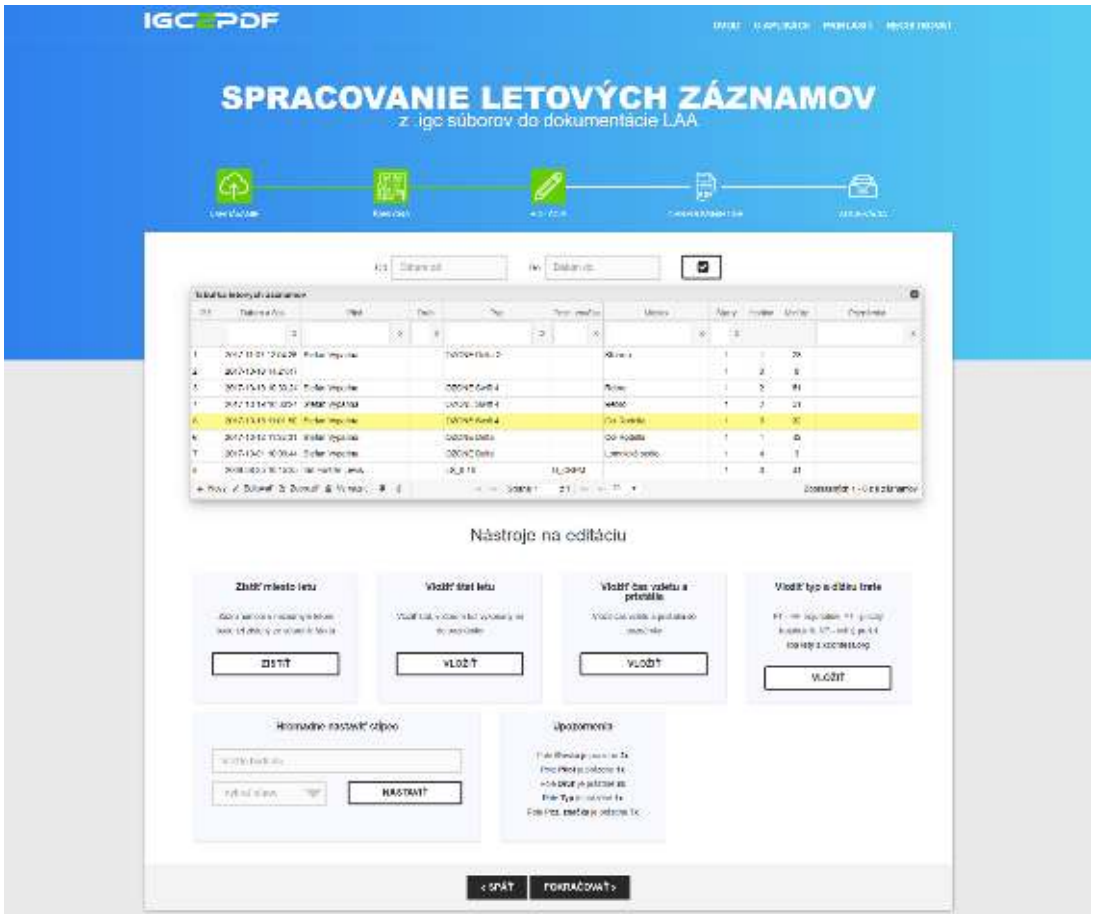
Účelom tejto akcie je nastaviť zvolenú šablónu záznamníka a taktiež uloženie zvoleného letového športového zariadenia. Model dátového toku je znázornený na



Obr. 30: *DataFlowDiagram* akcie *SetTemplateAction* kontroléra *Templates*

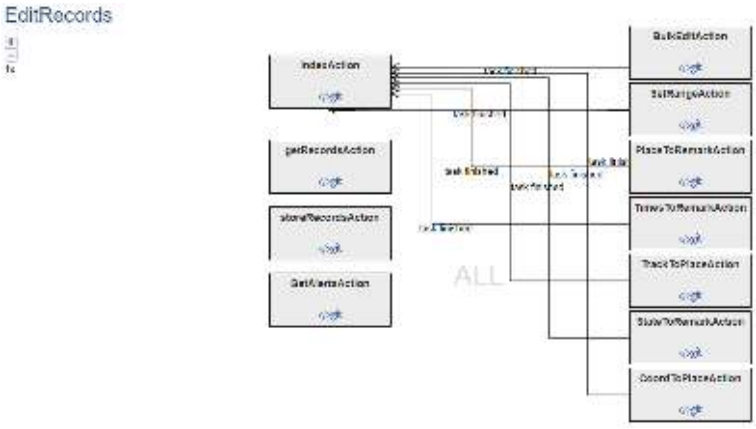
4.3.3. EditRecords – editovanie letových záznamov

Tento kontrolér zapuzdruje všetku funkcionalitu, ktorá umožňuje používateľovi editovať svoje letové záznamy. Na Obr. 31 sa nachádza pre lepšie predstavenie snímka obrazovky tohto kontroléra.



Obr. 31: 3. stav – obrazovka editácie letových záznamov

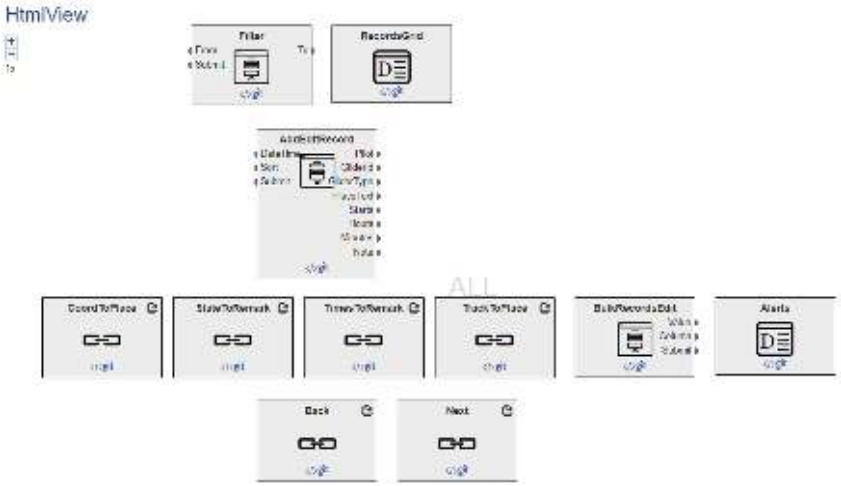
Tento kontrolér je najrozsiahlejším v aplikácií. Model jeho akcií sa nachádza na Obr. 32



Obr. 32: Akcie kontroléra *EditRecords*

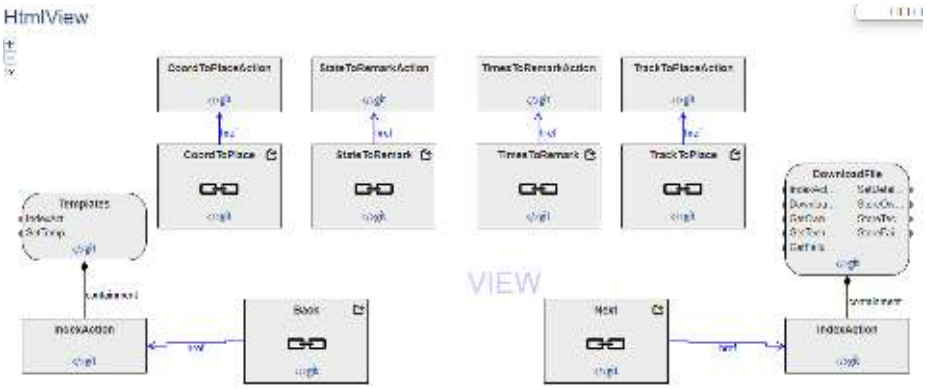
4.3.3.1. IndexAction – vykreslenie šablóny editácie letových záznamov

Model pohľadu zo snímky obrazovky na Obr. 31 sa nachádza na Obr. 33.



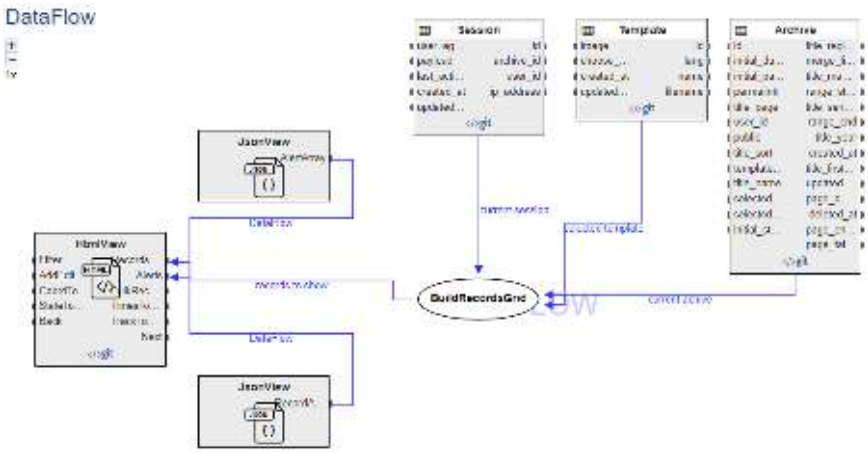
Obr. 33: *ViewDiagram* akcie *IndexAction* kontoléru *EditRecords*

Po prepnutí do *crosscut* pohľadu vidíme model vizualizácie hypertextových odkazov na Obr. 34.



Obr. 34: Model pre vizualizáciu hypertextových odkazov akcie *IndexAction* kontroléra *EditRecords*

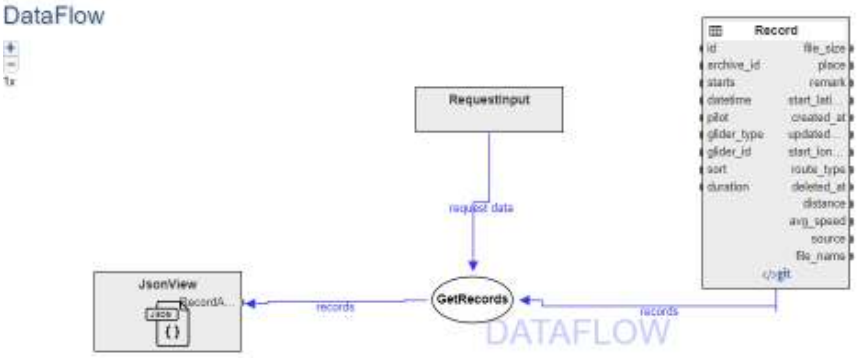
Diagram dátových tokov tejto akcie sa nachádza na Obr. 35.



Obr. 35: *DataFlowDiagram* akcie *IndexAction* kontoléra *EditRecords*

4.3.3.2. GetRecordsAction – získanie letových záznamov do data-gridu

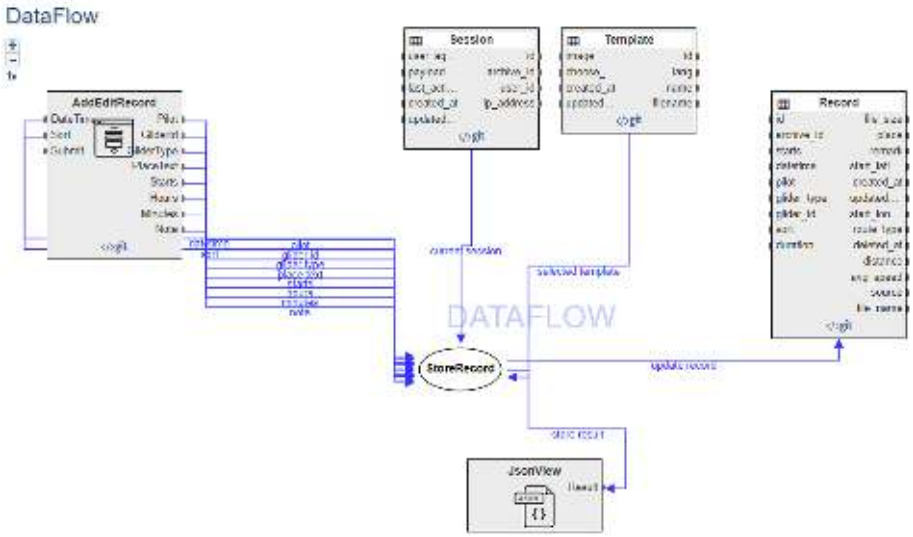
Táto akcia ma na starosti zabezpečiť zdrojové dáta pre vykreslenie gridu obsahujúceho letové záznamy. Diagram pohľadu nebudeme špeciálne uvádzať, pretože je triviálny a obsiahnutý v diagrame dátového toku tejto akcie zobrazeného na Obr. 36.



Obr. 36: *DataFlowDiagram* akcie *GetRecordsAction* kontroléra *EditRecords*

4.3.3.3. StoreRecordsAction – perzistovanie letového záznamu

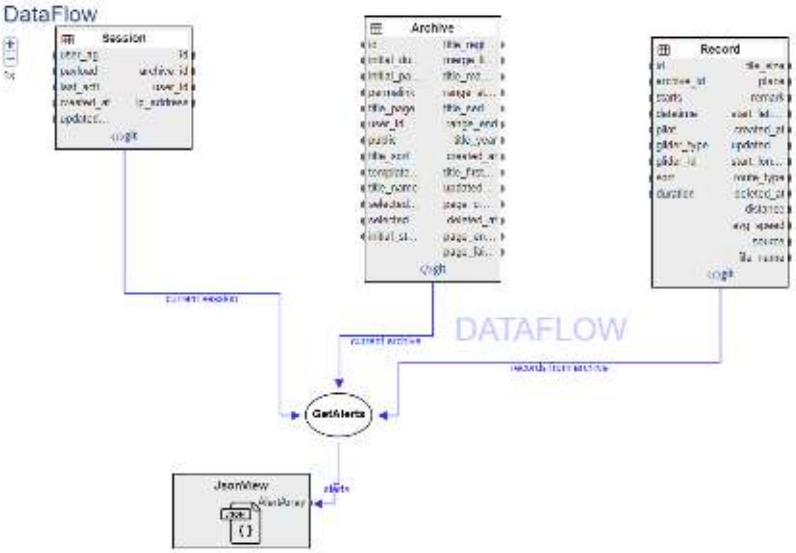
Táto akcia je zodpovedná za proces vytvárania a ukladania letového záznamu editovaného používateľom. Informácia o výsledku operácie je odosielaná ako *JsonView* a znova je súčasťou diagramu dátového toku zobrazeného na Obr. 37.



Obr. 37: *DataFlowDiagram* akcie *StoreRecordsAction* kontroléru *EditRecords*

4.3.3.4. GetAlertsAction – získanie upozornení ohľadom letových záznamov

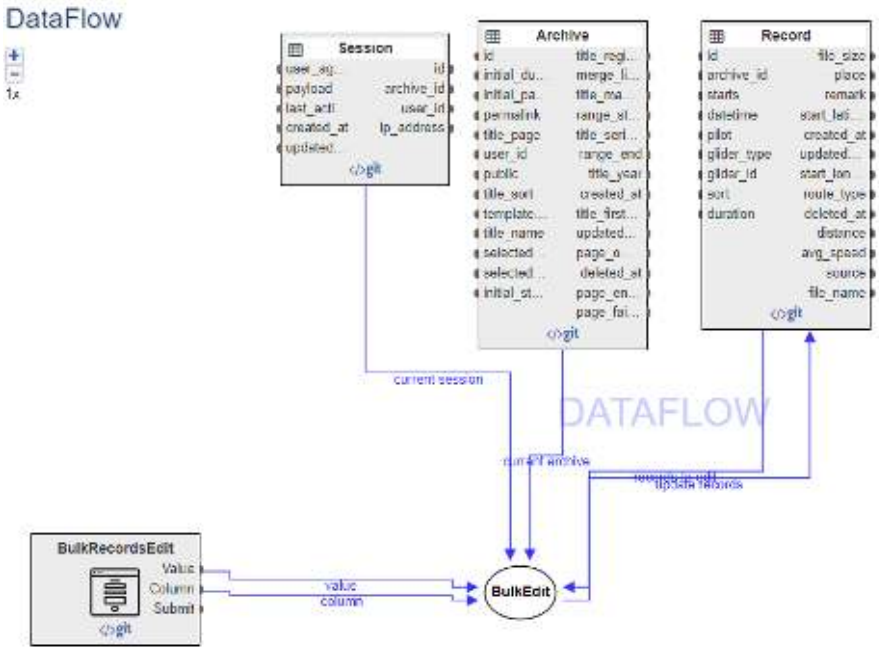
Ako je možné vidieť na snímke obrazovky na Obr. 31, tak na tejto obrazovke sa nachádza blok s upozorneniami ohľadom letových záznamov. Diagram dátového toku tejto akcie je znázornený na Obr. 38.



Obr. 38: *DataFlowDiagram* akcie *GetAlertsAction* kontroléru *EditRecords*

4.3.3.5. BulkEditAction – hromadná úprava záznamov

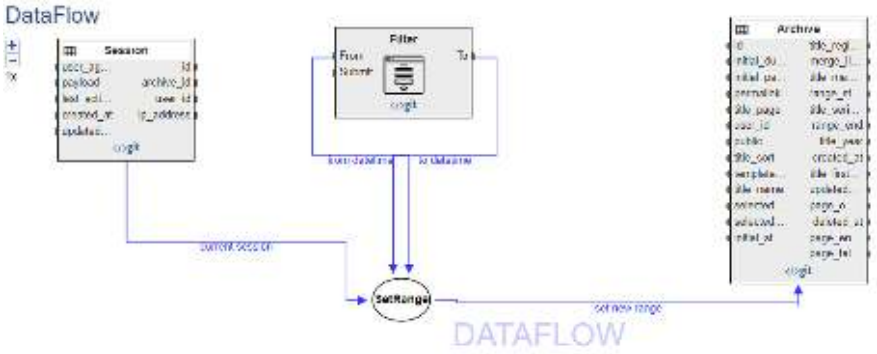
Táto akcia zodpovedá za proces hromadnej úpravy letových záznamovo. Nemá žiadnu formu pohľadu, pretože po jej ukončení je stav aplikácie presmerovaný do akcie *IndexAction* kontroléra *EditRecords*. To isté platí aj o všetkých zvyšných 6 akciách tohto kontroléra, ktoré budú obsahom ďalších podkapitol. Diagram dátového toku tejto akcie je zobrazený na Obr. 39.



Obr. 39: *DataFlowDiagram* akcie *BulkEditAction* kontroléra *EditRecords*

4.3.3.6. SetRangeAction – nastavenie časového rozsahu

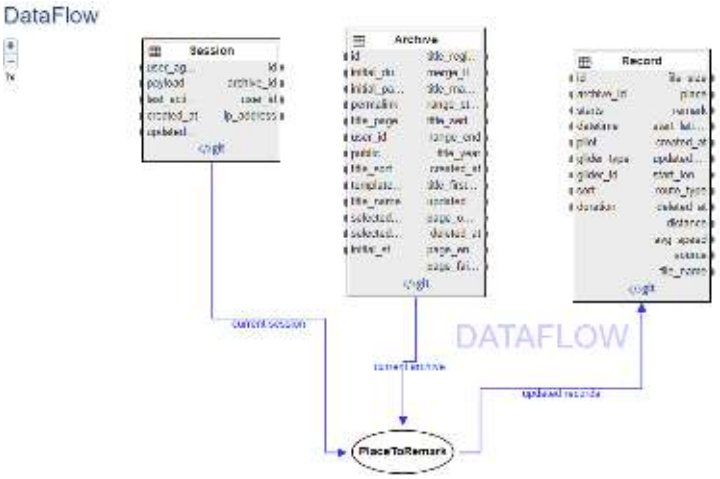
Táto akcia predstavuje proces nastavenia časového rozsahu pre zobrazované letové záznamy. Dátový tok je namodelovaný na Obr. 40.



Obr. 40: *DataFlowDiagram* akcie *SetRangeAction* kontroléra *EditRecords*

4.3.3.7. PlaceToRemarkAction – nastavenie miesta štartu do poznámky

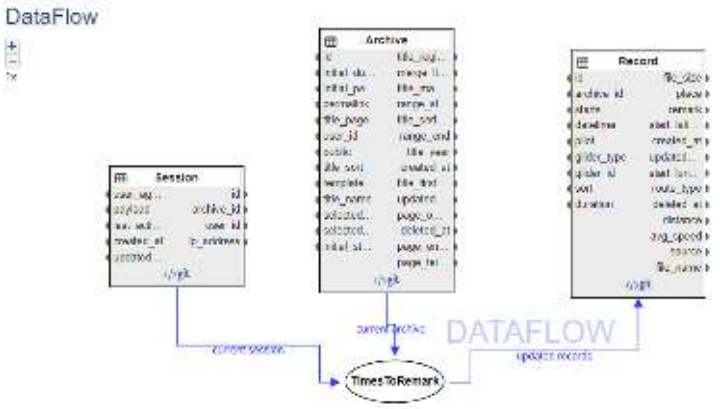
Táto akcia zabezpečuje nastavenie miesta štartu do poznámok jednotlivých letových záznamov. Dátový tok je zobrazený na Obr. 41.



Obr. 41: *DataFlowDiagram* akcie *PlaceToRemarkAction* kontroléra *EditRecords*

4.3.3.8. TimesToRemarkAction – čas vzletu a pristátia do poznámky

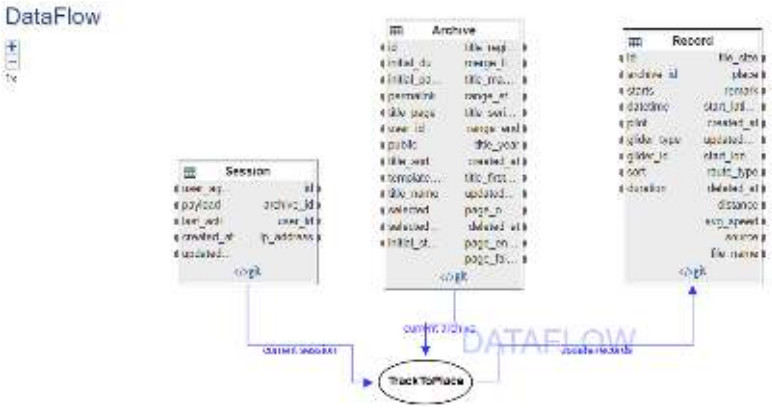
V tejto akcii sa rieši pridávanie času vzletu a pristátia do poznámky pre jednotlivé letové záznamy. Diagram dátového toku je zobrazený na Obr. 42.



Obr. 42: *DataFlowDiagram* akcie *TimesToRemarkAction* kontroléra *EditRecords*

4.3.3.9. TrackToPlaceAction – typ a dĺžka trate do poznámky

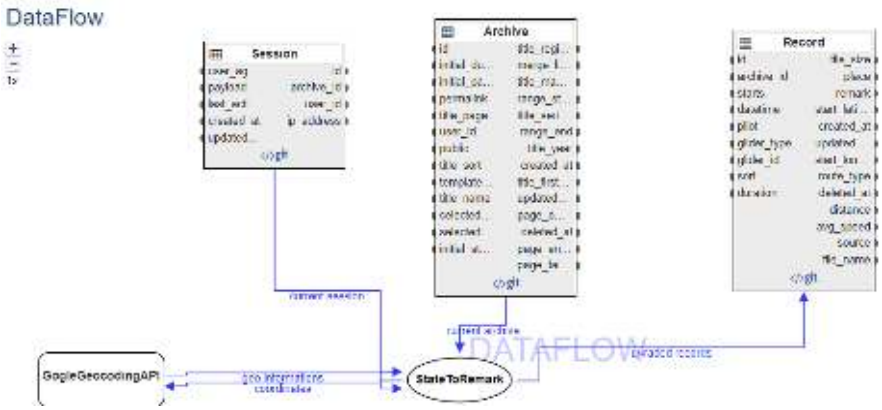
Úlohou tejto akcie je vložiť typ a dĺžku trate do poznámky jednotlivých letových záznamov. Dátový tok je namodelovaný na Obr. 43.



Obr. 43: *DataFlowDiagram* akcie *TrackToPlaceAction* kontroléra *EditRecords*

4.3.3.10. StateToRemarkAction – štát do poznámky

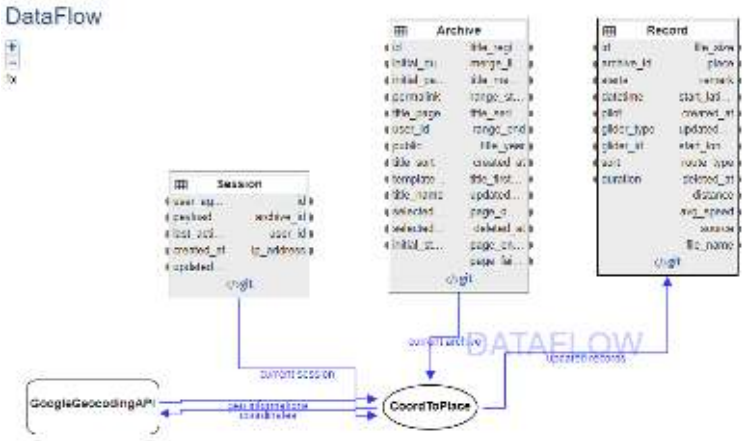
V tejto akcii je identifikovaný štát vzletu a uložený do poznámky pre jednotlivé letové záznamy. Diagram dátového toku je namodelovaný na Obr. 44.



Obr. 44: *DataFlowDiag*ram akcie *StateToRemarkAction* kontroléra *EditRecords*

4.3.3.11. CoordToPlace – zistenie miesta vzletu

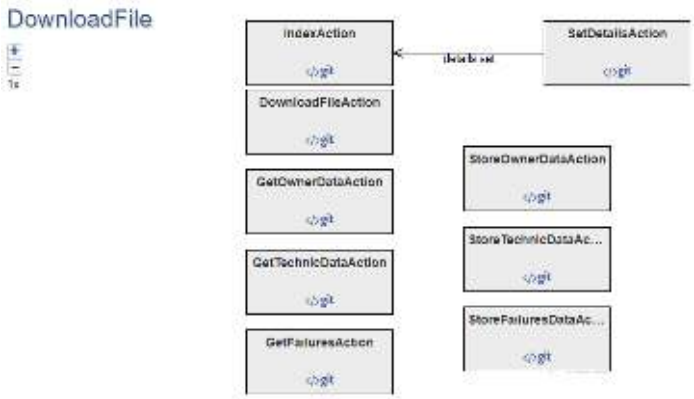
V tejto akcii je zistené miesto vzletu na základe súradníc pri vzlete. Model diagramu dátového toku je zobrazený na Obr. 45.



Obr. 45: *DataFlowDiagram* akcie *CoordToPlaceAction* kontoléra *EditRecords*

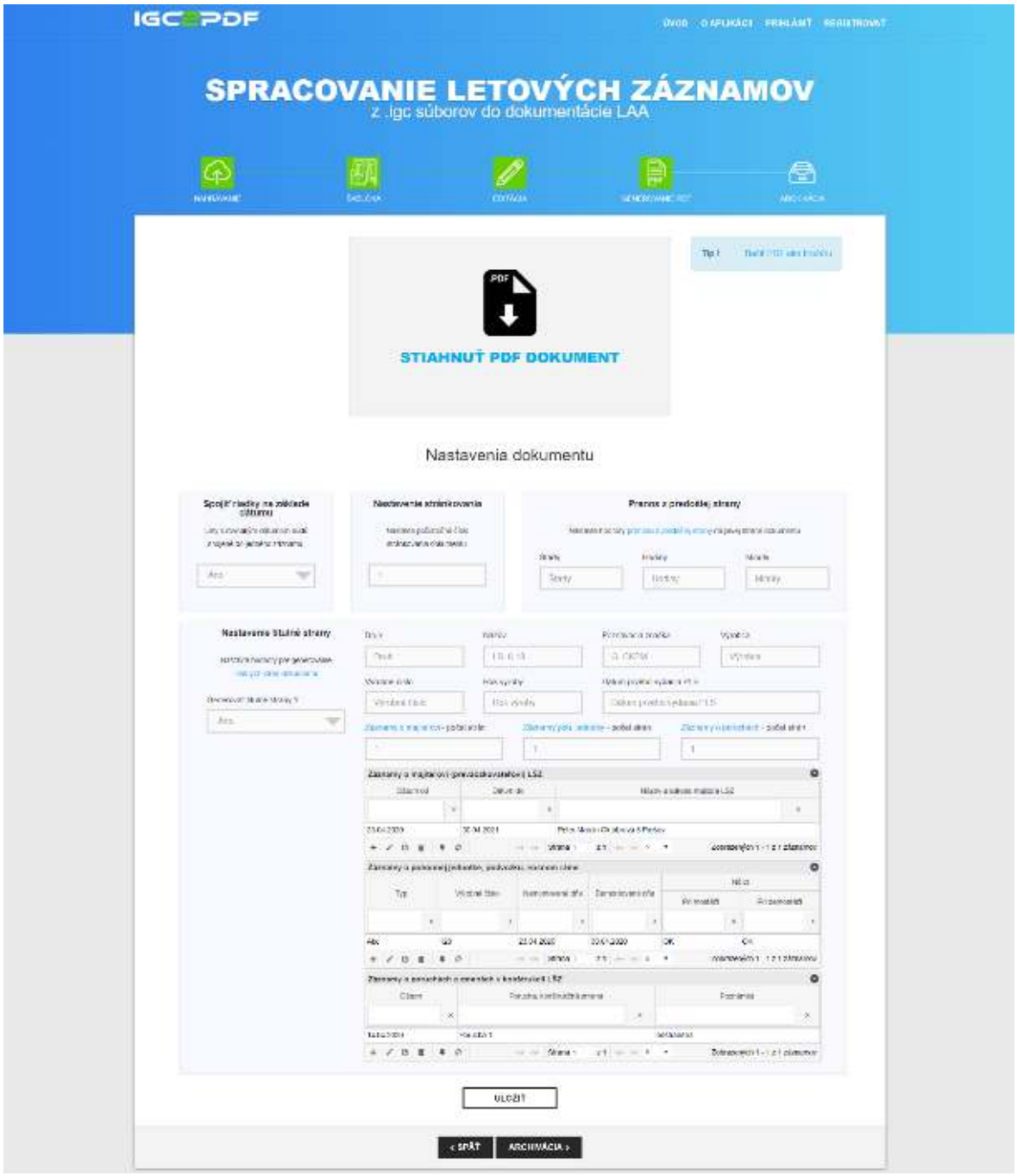
4.3.4. DownloadFile – stiahnutie vygenerovaného záznamníka

Tento kontrolér združuje funkcionalitu okolo generovania, sťahovania a nastavovania detailov letového záznamníka. Jeho akcie sú namodelované na Obr. 46.



Obr. 46: Akcie kontroléra *DownloadFile*

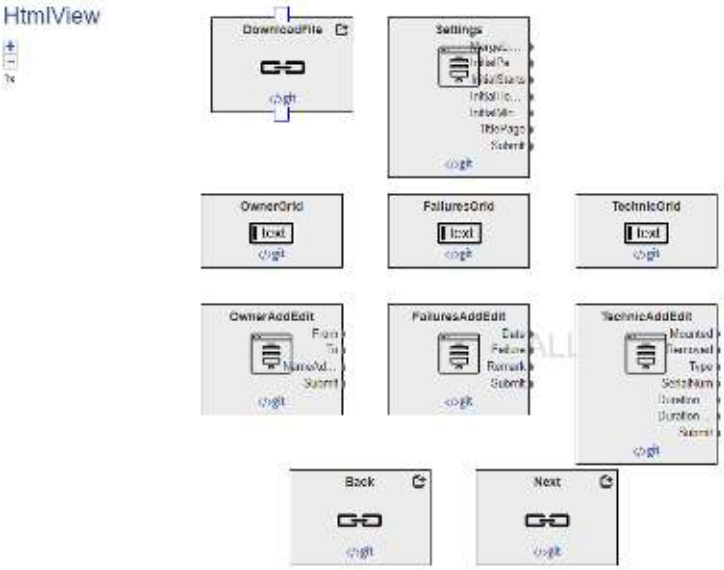
Kontrolér predstavuje taktiež ďalší krok v systéme, ktorý má aj svoju obrazovku zachytenú na Obr. 47.



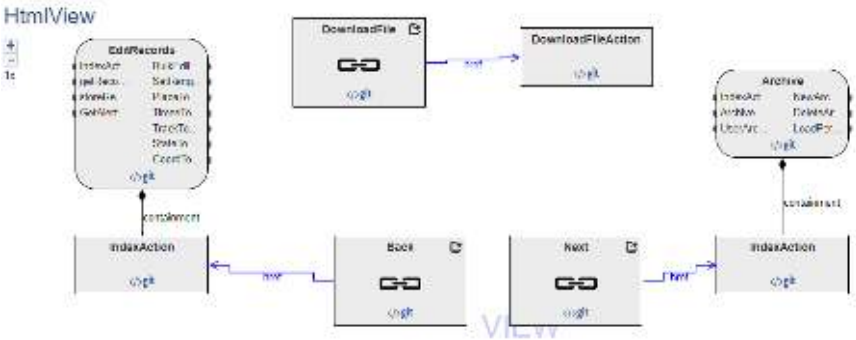
Obr. 47: 4. stav – nastavenia a stiahnutie letovej dokumentácie

4.3.4.1. IndexAction – vykreslenie šablóny

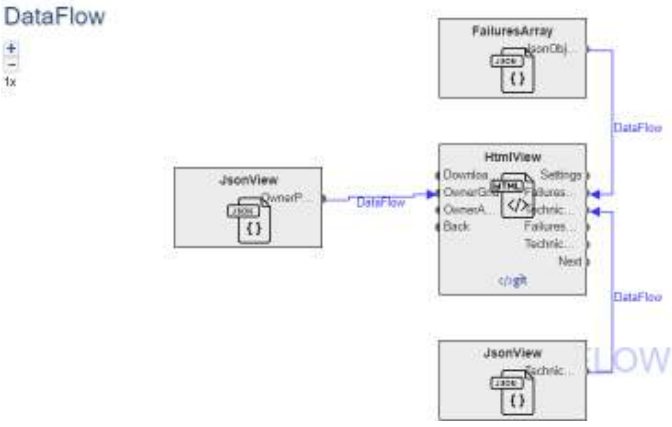
Táto akcia je primárne určená pre vykreslenie obrazovky zachytenej na Obr. 47. Diagram pohľadu tejto obrazovky sa nachádza na Obr. 48. Pri prepnutí sa do *crosscut* Obr. 49 pohľadu vidíme model zachytávajúci hypertextové prepojenia. Na záver uvádzame aj diagram dátového toku namodelovaného na Obr. 50.



Obr. 48: *ViewDiagram* akcie *IndexAction* kontroléra *DownloadFileController*



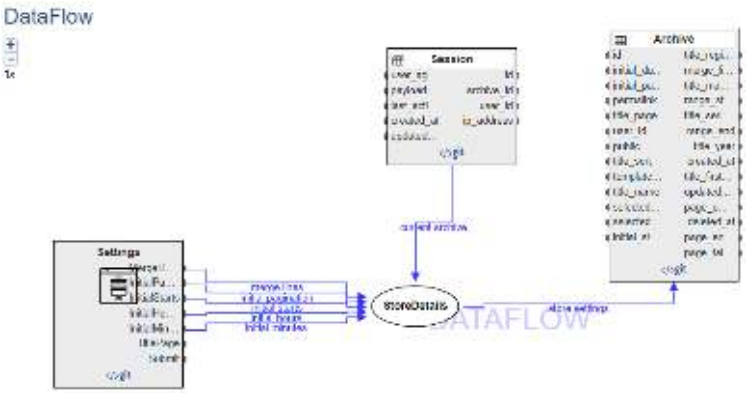
Obr. 49: Hypertextové prepojenia akcie *IndexAction* kontroléra *DownloadFileController*



Obr. 50: *DataFlowDiagram* akcie *IndexAction* kontroléra *DownloadFile*

4.3.4.2. SetDetailsAction – nastavenia dokumentov

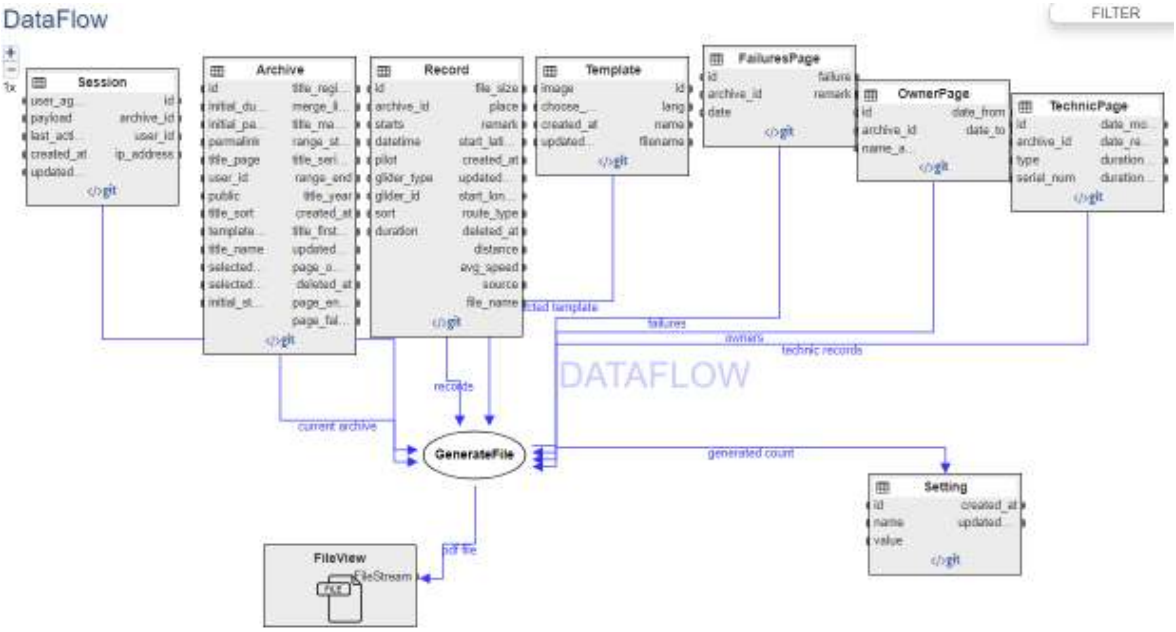
Táto akcia má na starosti perzistovať nastavenia dokumentov záznamníka. Jeho dátový tok je namodelovaný na Obr. 51.



Obr. 51: *DataFlowDiagram* akcie *SetDetailsAction* kontroléra *DownloadFile*

4.3.4.3. DownloadFileAction – vygenerovanie PDF záznamníka

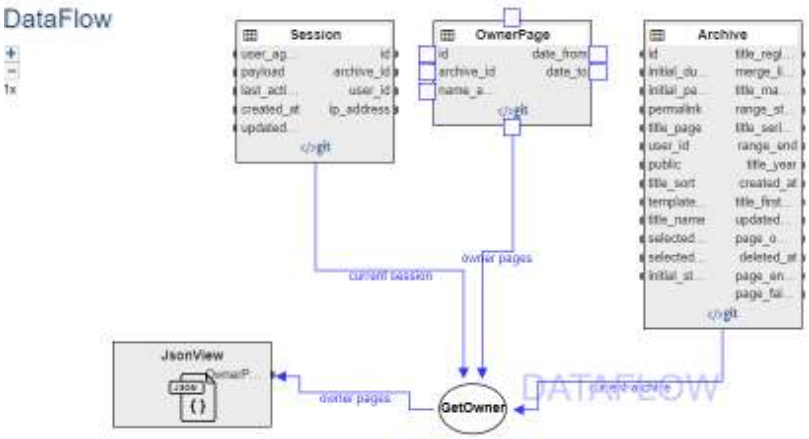
Účelom tejto akcie je vygenerovanie kompletného záznamníka do súboru PDF. Dátový tok je namodelovaný na Obr. 52.



Obr. 52: *DataFlowDiagram* akcie *DownloadFileAction* kontroléra *DownloadFile*

4.3.4.4. GetOwnerDataAction – získanie dát o majiteľovi LŠZ

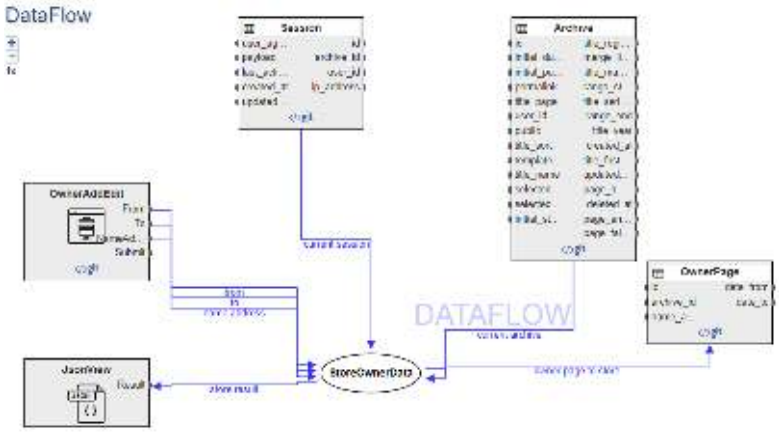
Táto akcia poskytuje dáta o majiteľovi LŠZ. Diagram dátového toku je namodelovaný na Obr. 53.



Obr. 53: *DataFlowDiagram* akcie *GetOwnerDataAction* kontroléra *DownloadFile*

4.3.4.5. StoreOwnerDataAction – uloženie dát o majiteľovi LŠZ

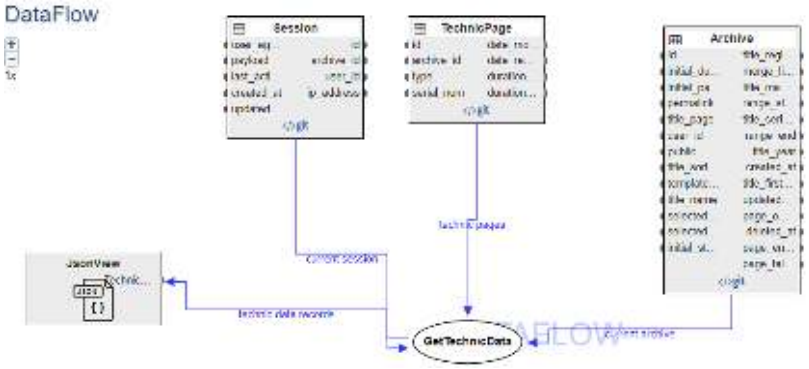
Táto akcia perzistuje dáta o majiteľovi LŠZ. Jej model toku dát je na Obr. 54.



Obr. 54: *DataFlowDiagram* akcie *StoreOwnerDataAction* kontroléra *DownloadFile*

4.3.4.6. GetTechnicDataAction – získanie technických dát LŠZ

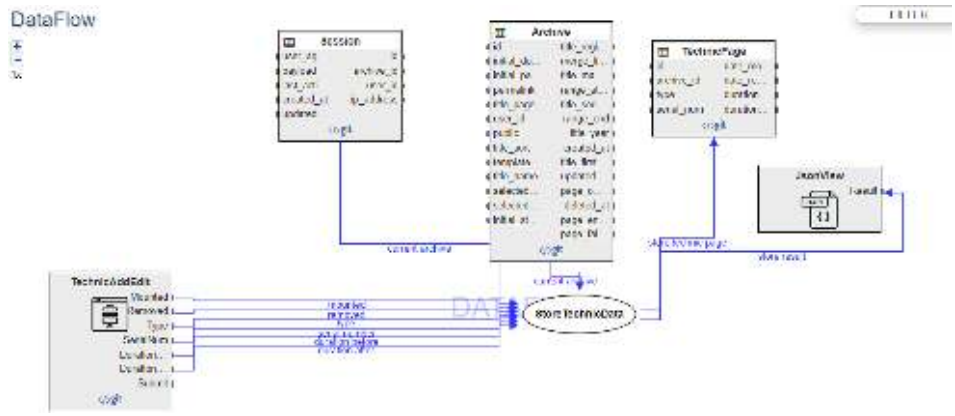
Táto akcia poskytuje technické dáta pre LŠZ. Dátový tok je namodelovaný na Obr. 55.



Obr. 55: *DataFlowDiagram* akcie *GetTechnicDataAction* kontroléra *DownloadFile*

4.3.4.7. StoreTechnicDataAction – perzistovanie technických dát LŠZ

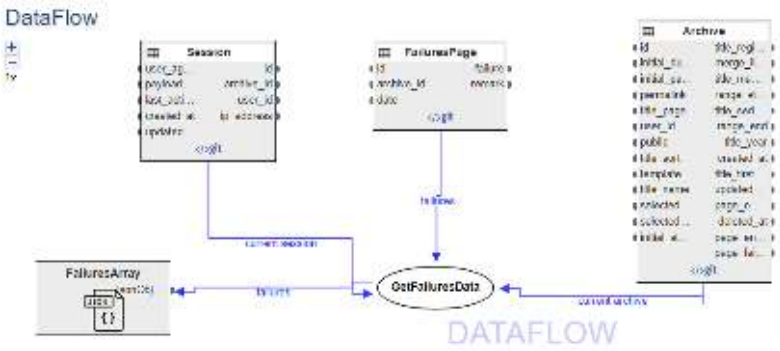
Táto akcia poskytuje perzistovanie technických dát pre LŠZ. Dátový tok je namodelovaný na Obr. 56.



Obr. 56: *DataFlowDiagram* akcie *StoreTechnicDataAction* kontroléra *DownloadFile*

4.3.4.8. GetFailuresAction – získanie dáta o poruchách

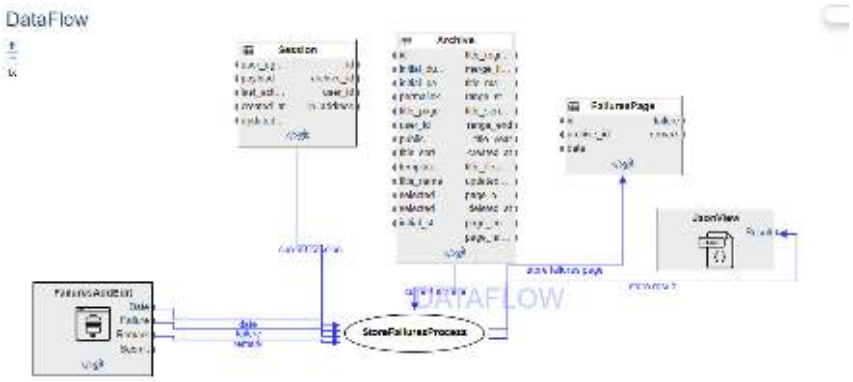
Táto akcia vracia dáta o poruchách. Diagram dátového toku je namodelovaný na Obr. 57.



Obr. 57: *DataFlowDiag*ram akcie *GetFailuresAction* kontroléra *DownloadFile*

4.3.4.9. StoreFailuresAction – perzistovanie porúch

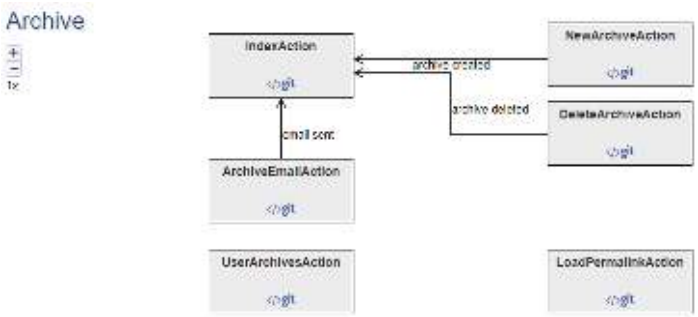
Táto akcia má na starosti perzistovanie dát ohľadom porúch LŠZ. Jej model dátového toku je namodelovaný na Obr. 58.



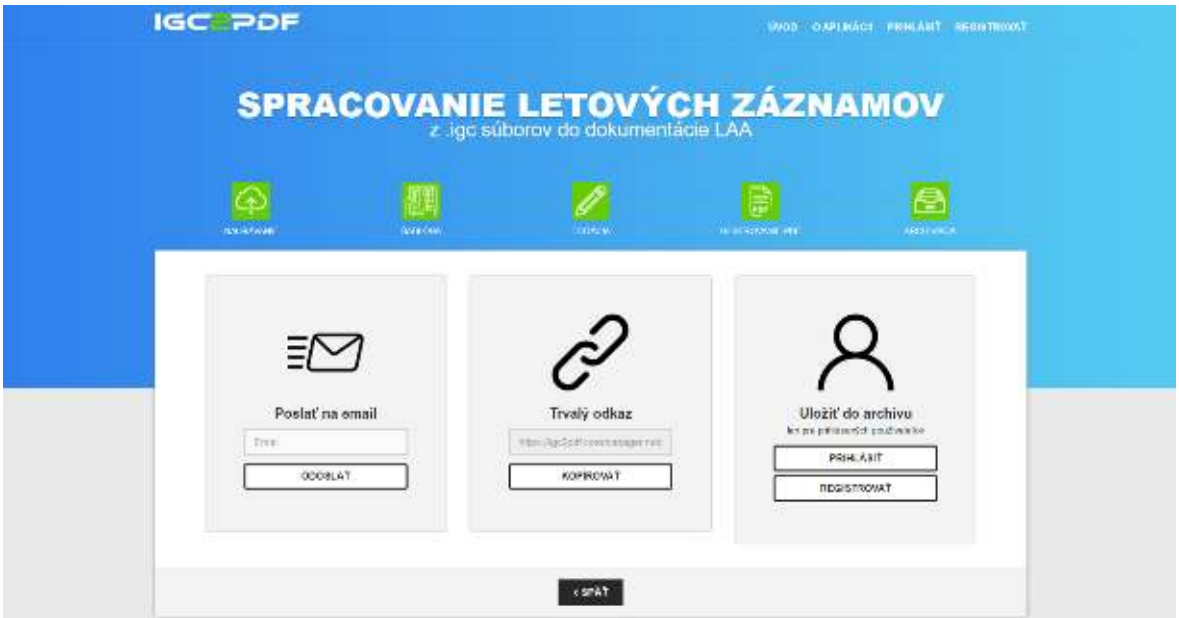
Obr. 58: *DataFlowDiag*ram akcie *StoreFailuresAction* kontroléra *DownloadFile*

4.3.5.Archive – archivácia práce

Tento kontrolér zlučuje celú funkcionalitu ohľadom možností archivácie práce. Jeho akcie sú znázornené na Obr. 59. Pre lepšie porozumenie je na Obr. 60 zobrazená snímka obrazovky tohto kontroléra.



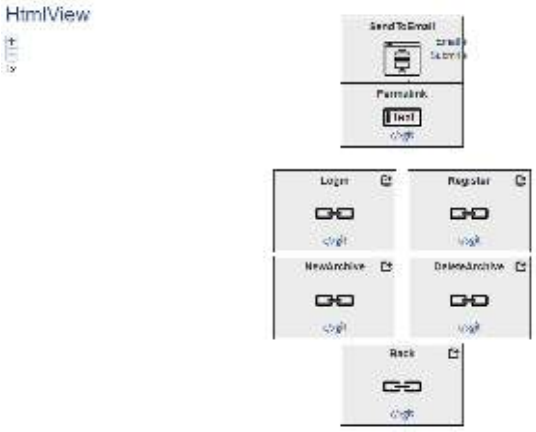
Obr. 59: Akcie kontroléra *Archive*



Obr. 60: 5. stav – obrazovka archivácia práce

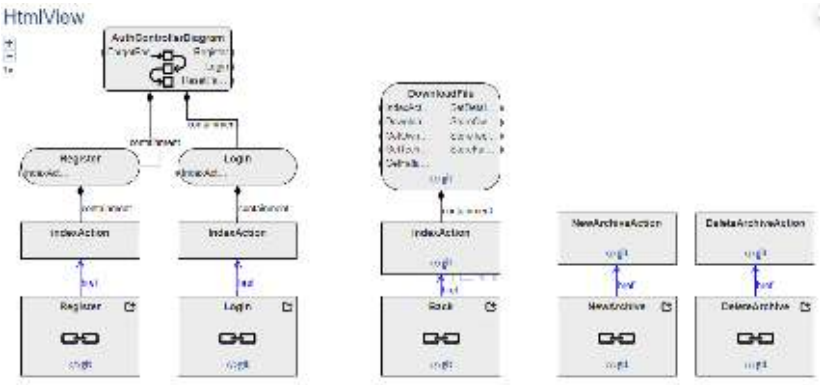
4.3.5.1. IndexAction – zobrazenie stránky archivácie

Účelom tejto akcie je zobrazenie stránky archivácie. Diagram pohľadu je zobrazený na Obr. 61.



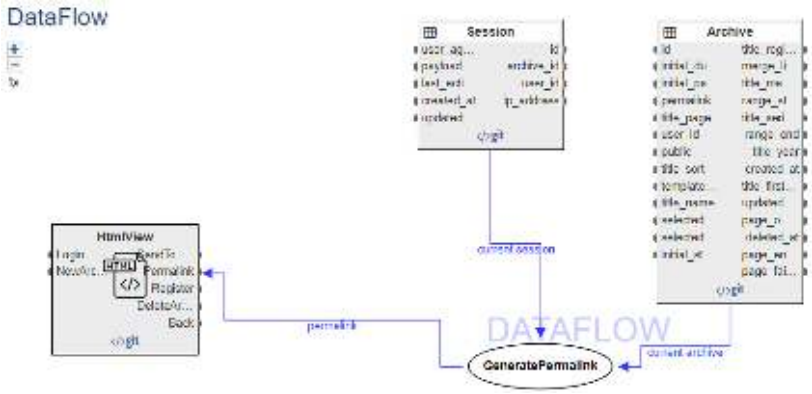
Obr. 61: Diagram pohľadu pre šablónu archivácie kontroléra *Archive*

Zobrazenie *crosscut* zobrazuje model hypertextového prepojenia na Obr. 62.



Obr. 62: Diagram hypertextových prepojení kontroléra *Archive*

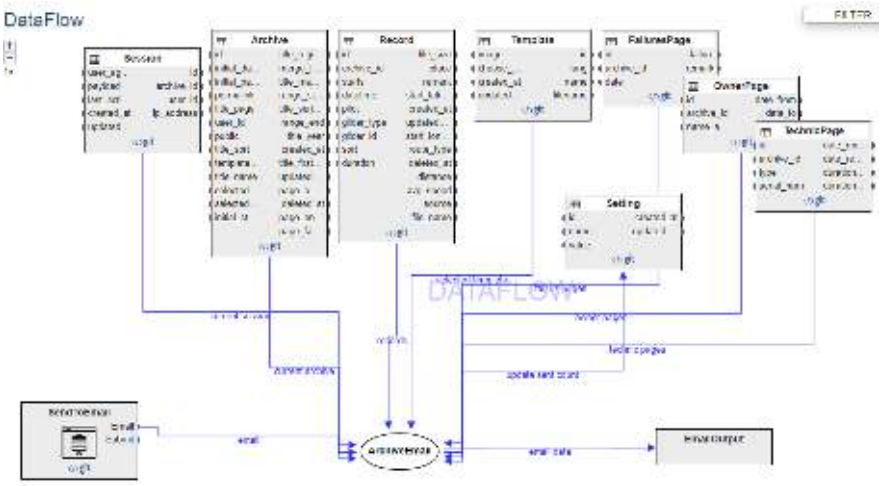
Diagram dátového toku tejto akcie je zobrazený na Obr. 63.



Obr. 63: *DataFlowDiag*ram akcie *IndexAction* kontroléra *Archive*

4.3.5.2. ArchiveEmailAction – archivácia na email

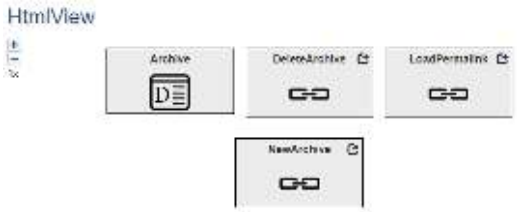
Táto akcia má na starosti archivovanie práce na email s PDF prílohou záznamníka. Dátový tok je namodelovaný na Obr. 64.



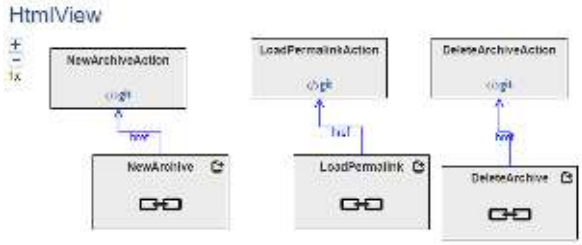
Obr. 64: *DataFlow* akcie *ArchiveEmailAction* kontroléra *Archive*

4.3.5.3. UserArchivesAction – zoznam archívov používateľa

Táto akcia obsahuje zoznam archívov používateľa. Jeho diagram pohľadu je na Obr. 65 spolu s diagramom hypertextového prepojenia na Obr. 66.

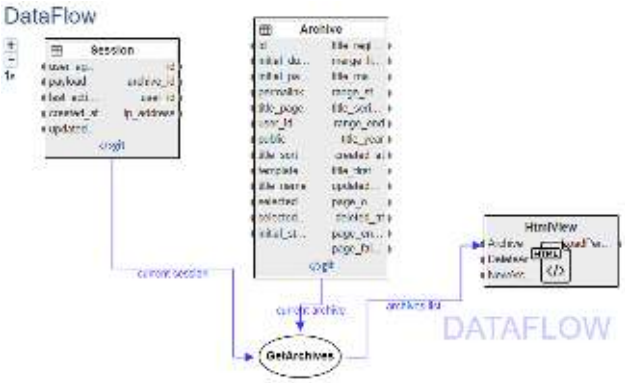


Obr. 65: Diagram pohľadu akcie *ArchiveEmailAction* kontroléra *Archive*



Obr. 66: Diagram hypertextového prepojenia akcie *ArchiveEmailAction* kontroléra *Archive*

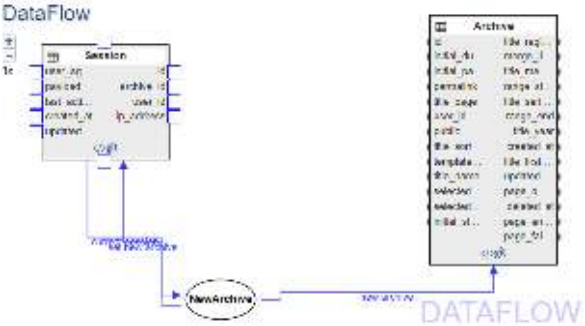
Model dátového toku je namodelovaný na Obr. 67.



Obr. 67: *DataFlow* akcie *ArchiveEmailAction* kontroléra *Archive*

4.3.5.4. NewArchiveAction – vytvorenie nového archívu

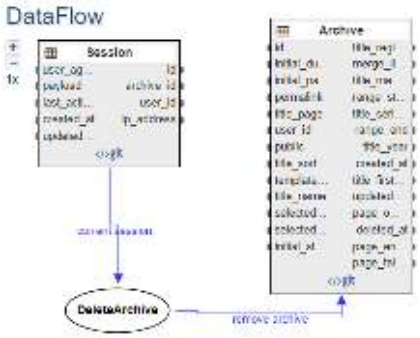
Táto akcia má na starosti vytvoriť nový archív. Jej model dátového toku je na Obr. 68.



Obr. 68: *DataFlow* akcie *NewArchiveAction* kontroléra *Archive*

4.3.5.5. DeleteArchiveAction – vymazanie archívu

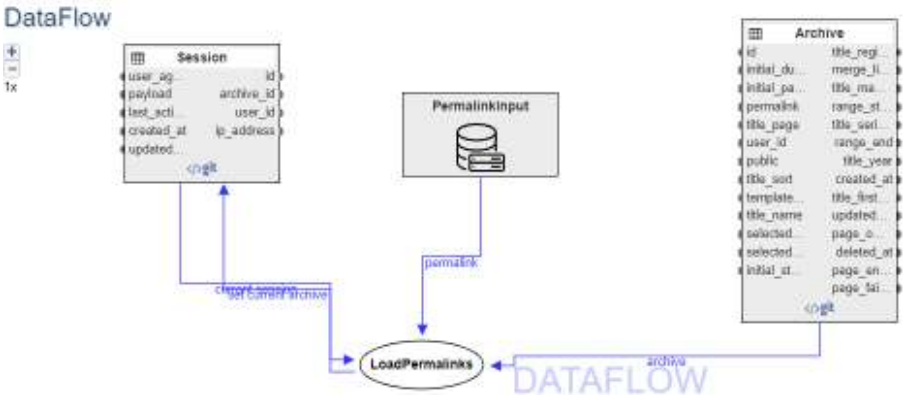
O vymazanie archívu sa stará táto akcia. Jej dátový tok je namodelovaný na Obr. 69.



Obr. 69: *DataFlow* akcie *DeleteArchiveAction* kontroléra *Archive*

4.3.5.6. LoadPermalinkAction – načítanie archívu z trvalého odkazu

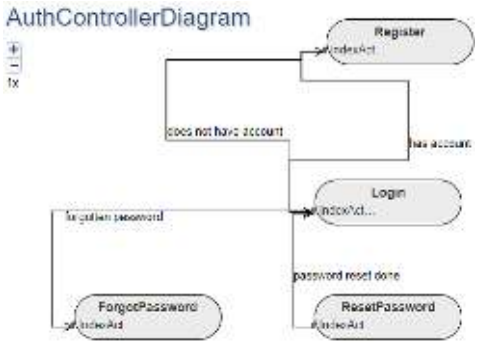
Posledná akcia tohto kontroléra má na starosti načítanie archívu na základe trvalého odkazu obsiahnutého v URL adrese. Jeho dátový tok sa nachádza na Obr. 70.



Obr. 70: *DataFlow* akcie *LoadPermalinkAction* kontroléra *Archive*

4.4. AuthControllerDiagram – modul autentifikácie

Tento modul má na starosti rutiny týkajúce sa autentifikácie, ktorými sú registrácia, prihlásenie, funkcia zabudnutého hesla spolu so zmenou tohto hesla. Stavy tohto modulu sú namodelované pomocou *ControllerDiagramu* zobrazeného na Obr. 71.



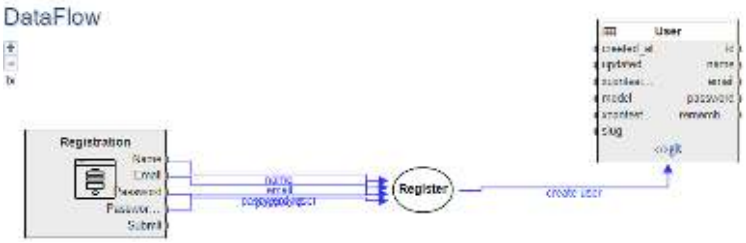
Obr. 71: *AuthControllerDiagram*

4.4.1. Register – registrácia

Tento kontrolér zabezpečuje službu registrácie. Obsahuje jedinú akciu *IndexAction*, ktorej šablóna je namodelovaná cez *ViewDiagram* namodelovaný na Obr. 72.

Obr. 72: *ViewDiagram* kontroléra *Register*

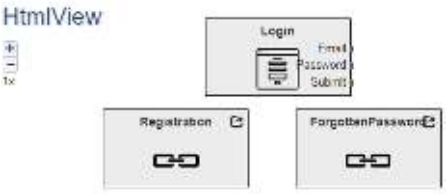
Dátový tok je namodelovaný na Obr. 73.



Obr. 73: *DataFlow* kontroléra *Register*

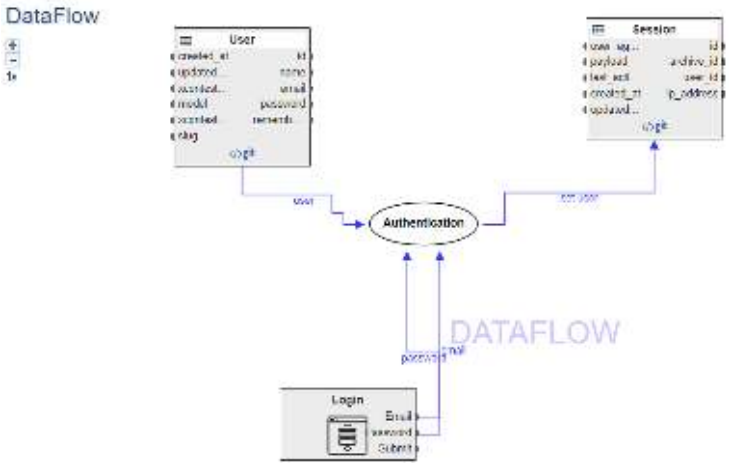
4.4.2. Login – prihlásenie

Logiku autentifikácie má na starosti tento kontrolér. Taktiež obsahuje iba jednu akciu *IndexAction*. Jeho šablóna je namodelovaná na Obr. 74.



Obr. 74: *ViewDiagram* kontroléra *Login*

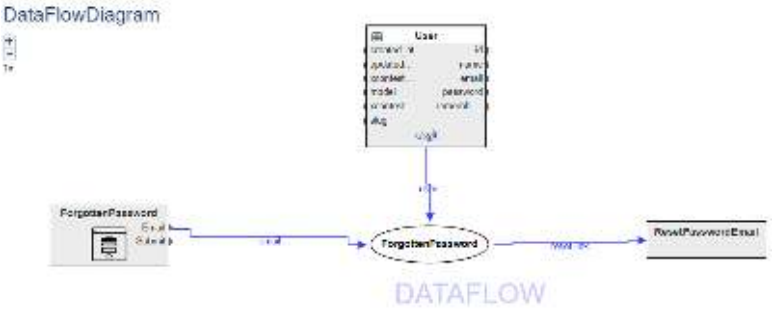
Dátový tok prihlásenia je namodelovaný na Obr. 75.



Obr. 75: *DataFlow* kontroléra *Login*

4.4.3. ForgottenPassword – vyžiadanie zmeny hesla

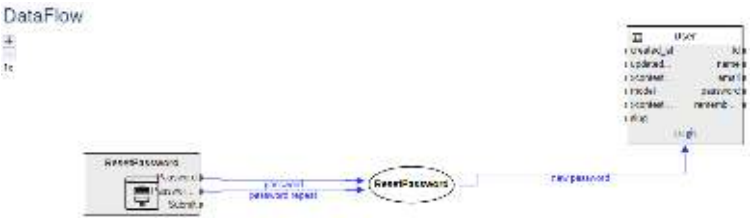
Proces vyžiadania zmeny hesla má na starosti tento kontrolér, ktorého dátový tok je namodelovaný na Obr. 76. Zároveň je v ňom zachytený aj model pohľadu.



Obr. 76: *DataFlow* diagram kontroléra *ForgottenPassword*

4.4.4. ResetPassword – zmena hesla

Tento kontrolér zabezpečuje zmenu používateľského hesla. Jeho diagram toku údajov je zobrazený na Obr. 77. Taktiež je na ňom zobrazený diagram šablóny.

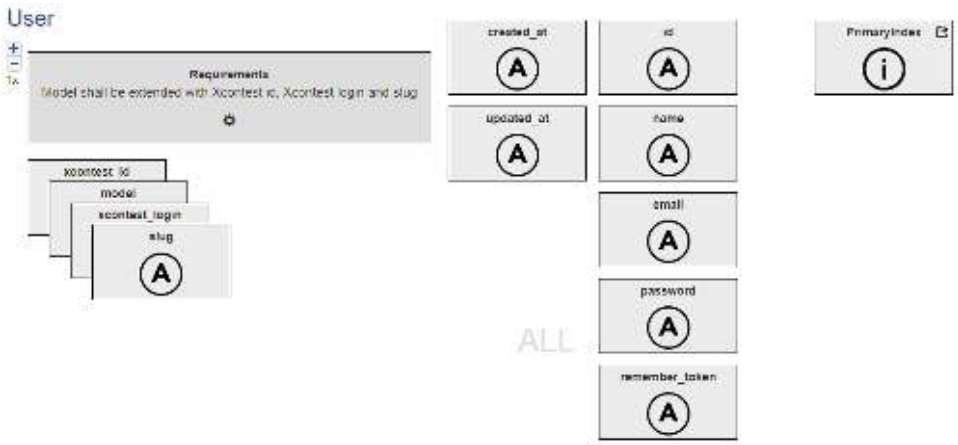


Obr. 77: *DataFlowDiagram* kontroléra *ResetPassword*

4.5. Ukážka generovania artefaktov z textových požiadaviek

Pre potreby ukážky generovania modelov z textových požiadaviek sme vybrali databázovú entitu *User* reprezentujúcu zaregistrovaného používateľa aplikácie.

Ako je možné vidieť na Obr. 78 objekt *Requirements* obsahuje zmenovú požiadavku v textovej podobe. Konkrétne sa jedna o požiadavku, aby pri používateľovi bolo možné uchovávať jeho identifikátory externej služby xcontest.org. Po spustení pluginu *ArtifactGenerator* sú požiadavky analyzované a na základe určitých pravidiel sú vytvorené nové objekty tohto modelu. V tomto prípade tieto uzly sú umiestnené pod požiadavkami. Jedná sa o *xcontest_id, model, xcontest_login* a *slug.* Vývojár teraz rozhodne čo s týmito objektmi je potrebné ďalej robiť. Objekt s názvom *model* v tomto kontexte nedáva žiadny zmysel, takže ho vývojár odstráni.



Obr. 78: Generovanie artefaktov entity *User.*

Zdroje

[1]. FERENČÍK FRANTIŠEK. Spracovanie letových záznamov v IGC formáte do textových dokumentov. Bakalárska práca. [Online] Technická univerzita v Košiciach, 25.05.2018. [cit. 2019-12-03]. Dostupné na internete: https://opac.crzp.sk/?fn=detailBiblioForm&sid=5301BA0C1F12FE31B5404AA915FD.

[2]. F. FERENČÍK. IGC2PDF Video ukážka, 2018, Dostupné online: < https://youtu.be/dP_jaGXk3vo>