CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	Web portal for visualizing data from the satellite instrument SATRAM / Timepix
Student:	Bc. Herbert Waage
Supervisor:	MSc. Benedikt Ludwig Bergmann
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2017/18

Instructions

The goal of the work is to design and create a web portal to display the data acquired by the SATRAM/Timepix onboard the ESA Proba-V satellite operating in Low Earth Orbit from 2013. The portal shall visualize data in standard web browsers with interactive graphics to show the radiation levels, the pixel matrices and the positions of the spacecraft in orbit. Data (raw frames) from the spacecraft are retrieved once a day, processed (frame/image analysis) and stored in ROOT files, in a database optimized for the Timepix applications at CERN. For SATRAM, slight modifications of the existing storage structure are needed.

The work includes:
1. Implementation of an automatic data preprocessing.
2. Creation of an efficient way for I/O operations to quickly scan huge datasets.
3. Design and implementation of a graphical user interface (GUI).
4. Definition of an application interface (API) for versatile query structure and integral data retrieval - this API will be used by GUI.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 20, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Master's thesis

Web portal for visualizing data from the satellite instrument SATRAM/Timepix

Bc. Herbert Waage

Supervisor: MSc. Benedikt Ludwig Bergmann

2nd May 2017

Acknowledgements

I would like to thank the thesis supervisor MSc. Benedikt Ludwig Bergmann for his time and willingness to provide me the necessary support in the matters related to physics. Another thank belongs to my family for supporting me during all my study. Last but not least thank to all the testers of the final application.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 2nd May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Waage, Herbert. *Web portal for visualizing data from the satellite instrument SATRAM/Timepix.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Tato práce se zabývá analýzou, návrhem, a implementací webové aplikace na zobrazování dat získaných SATRAM/Timepix detektorem umístěném na družici ESA Proba-V. Výsledkem je aplikace vizualizující pixelové matice (snímky), hladiny radiace, a pozice satelitu na orbitu. Vizualizase umožňuje také různé filtrování zobrazený dat.

Klíčová slova SATRAM, Timepix, pixelový detektor, detektor radiace, satelit, Proba-V, vizualizace

Abstract

This thesis deals with the analysis, design, and implementation of a web application to visualise the data acquired by the SATRAM/Timepix pixel detector onboard the ESA Proba-V satellite. The result of the thesis is the application showing the pixel matrices (frames), radiation levels, and positions of the spacecraft in the orbit. The visualisation provides various types of filtering the data.

Keywords SATRAM, Timepix, pixel radiation detector, satellite, Proba-V, visualisation

Contents

List of Figures

List of Tables

Introduction

Motivation

The lightweight SATRAM (Space Application of Timepix based Radiation Monitor) payload is operating in LEO orbit onboard the Proba-V satellite since May 2013 [1]. It provides high-resolution wide-range radiation monitoring of the environment [2] near the spacecraft. The payload is equipped with the Timepix detector that consists of a matrix of 256x256 pixels of pitch size 55µm for an area of 14x14mm.

Data from Timepix detector are transmitted, processed, and stored in the ROOT[1] file format developed for particle physics applications at CERN (European Organization for Nuclear Research)[2]. The goal of this thesis is to design and create a web portal for the data visualisation.

Thesis structure

The thesis is divided into following chapters:

- Chapter 1 "**Theoretical basis**" presents basic facts about Timepix detector, the payload carrying it and the radiation detection itself. It also describes the ROOT file format.

- Chapter 2 "**Analysis**" contains requirements, use cases, architecture scheme and domain model.

- Chapter 3 "**Design**" describes technologies used, algorithms, and methods. It contains also the graphical user interface design and relational database model.

[1] https://root.cern.ch/
[2] http://home.cern/

- Chapter 4 "**Realisation**" presents the crucial parts of the application implementation.
- Chapter 5 "**Testing**" consists of the heuristic evaluation and the user testing.

Chapter 1

Theoretical basis

This thesis is about visualising data from the Timepix detector. It is important to get to know the basic facts about this detector, the payload that is carrying it and the radiation detection itself. While I was creating this chapter I was using sources [1], [2], [3], [4], [5], [6], and [7].

1.1 Timepix

The Timepix is a hybrid semiconductor pixel detector of the Medipix family that was developed at CERN by the Medipix collaboration[3]. It is the successor of the successful Medipix2. Timepix provides high sensitivity, wide-dynamic range, high spatial resolution, and noiseless detection.

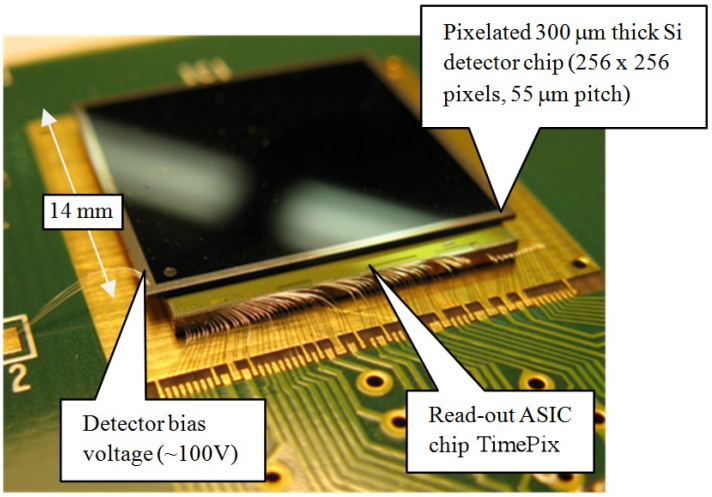

Figure 1.1: The Timepix detector; source: [5]

[3] http://medipix.web.cern.ch/medipix/

The device is equipped with a semiconductor detector (300 μm thick silicon) bump-bonded to a readout chip. The detector consists of 256x256 pixels (it is 65.536 total) of pitch size 55μm. All pixels cover the sensitive area of 14x14mm (1.98cm^2) - see figure 1.1.

1.1.1 Frames

The Timepix takes pictures of ionizing radiation (called frames). These frames will be visualised in the main area of the web portal. A typical frame acquired from the space environment is shown in the figure 1.2.

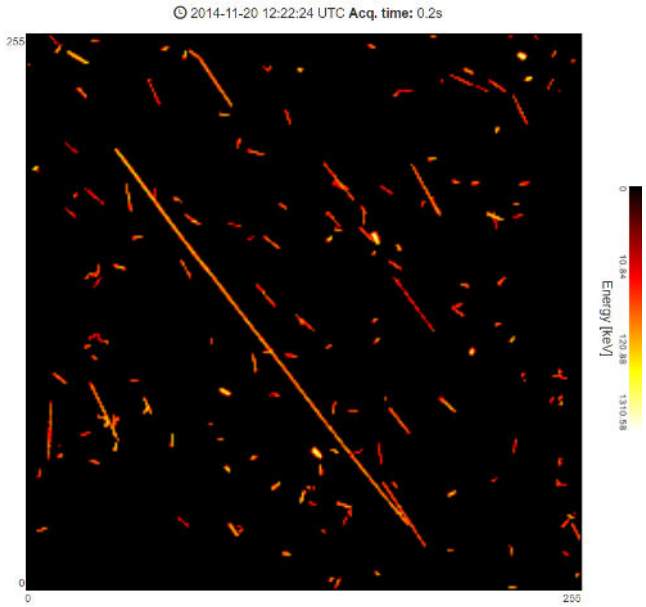

Figure 1.2: Particle tracks recorded by the Timepix detector. The acquisition time was 0.2s. The colour of each pixel represents the energy measured in it.

The pictures are taken by opening the detector shutter for all pixels at once. The pixel matrix is exposed for a time interval called the acquisition time or the frame exposure. This time usually varies from milliseconds to seconds. It depends on the incoming radiation. If there is a high radiation field, short acquisition times should be chosen (more frames per second) to not overexpose the picture. In low radiation fields the acquisition time should be longer (less frames per second) to reduce losses due to the dead time between the frames. The frame rate of the SATRAM payload (described in 1.2) is $\leq$ 5 frames per second.

After the frame is taken, it is stored in the form of a data packet for readout. The frame readout from the payload to the satellite computer takes about 100ms. It is influenced by the size of the packet that is given by the

pixel occupancy in the frame. The detector does not collect any data during this time.

1.1.2 Basic pattern recognition

After the frame data arrives from the satellite, the basic pattern recognition is done. The tracks are identified by a flood fill algorithm and divided into the 6 categories, which are shown in the figure 1.3: **dots** (only one-pixel), **small blobs** (round clusters with maximum 4 pixels), **heavy blobs** (round clusters with more than 5 pixels), **heavy tracks** (elliptical or straight clusters with a minimal width of three pixels), **straight tracks** (long straight clusters with more than 20 pixels length and less than three pixels width), and **curly tracks** (everything else).

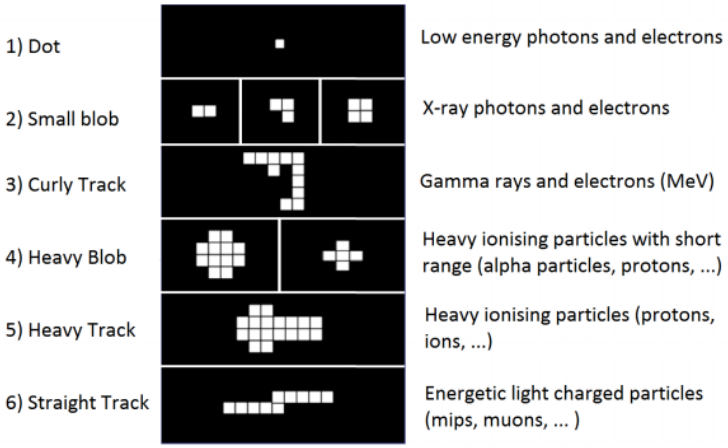

Figure 1.3: Track types; source: [7]

1.1.3 Timepix detector modes

Unlike Medipix2, Timepix provides extended functionality according to the level of per-pixel configuration. Each pixel in the Timepix detector can be configured to operate in one of the three modes illustrated in the figure 1.4.

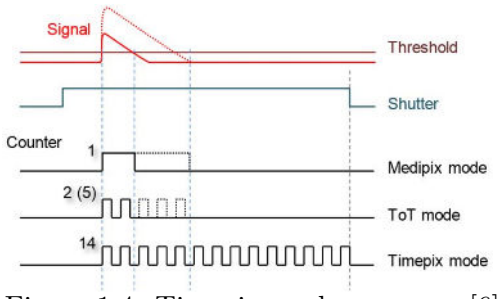

Figure 1.4: Timepix modes; source: [6]

- **Medipix mode** - Counts how often a pixel was triggered during the frame (more energy than 5keV - this is the "Threshold" in the picture).
- **Time-over-threshold (ToT) mode** - Counter allows the direct energy measurement in each pixel. This mode is used for all the frames I will be visualising in the final application.
- **Timepix mode** - Counter works as a timer and measures time of the particle detection. In other words it measures the Time-of-arrival of the particle - this is why this mode is also called ToA mode.

1.2 The SATRAM payload onboard ESA's Proba-V

The Timepix detector is in SATRAM payload on-board ESA's (European Space Agency) Proba-V satellite that is operating in LEO (Low-Earth Orbit) since May 2013. SATRAM means "The Space Application of Timepix Radiation Monitor" and it is a technology demonstration - the main idea of the project was to prove, that this technology can work in open space. There is FPGA in the payload that is controlling the Timepix detector and provides the communication with Proba-V, data compression and configuration. The alloy compartment is from Aluminium and the whole payload's dimensions are 108x63x56mm with it - see figure 1.5. The volume of the payload is 380ml and the weight is 172g. The entrance window in front of the Timepix chip is of 0.5mm thick Aluminium and the payload features 28V voltage input and power consumption $\leq$ 3W.

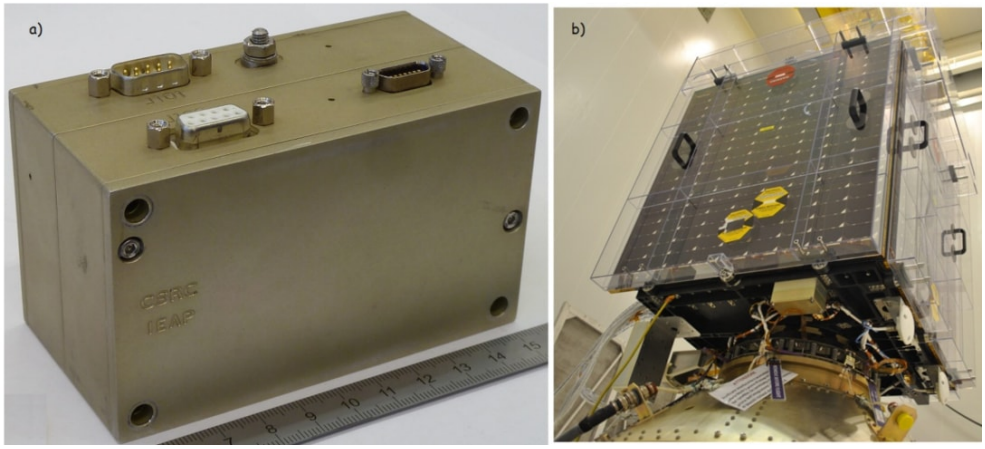

Figure 1.5: SATRAM payload on-board Proba-V satellite: a) SATRAM payload b) The payload attached to Proba-V satellite; source: [4]

The energy calibration of the Timepix chip was performed using X-rays of known energy. Test and response characterisation were performed by radi-

onuclide neutron source (AmBe) and low-energy light ion beams and mono-energetic fast neutron sources from the Van de Graaff accelerator at the IEAP (Institute of Experimental and Applied Physics) CTU in Prague.

1.2.1 Proba-V satellite position

Proba-V satellite is operating in LEO at approximately 820km altitude - see figure 1.6. For comparison the ISS (International Space Station) is working at 370km altitude.

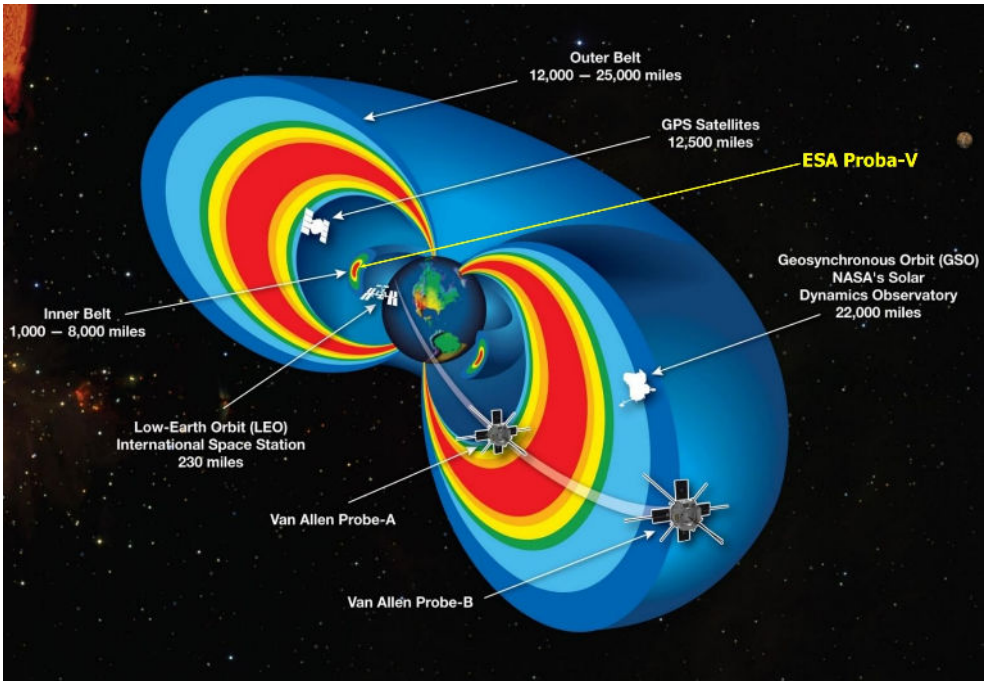

Figure 1.6: Proba-V satellite position; source: [6]

1.3 ROOT

The ROOT file format is used to store the data from the Timepix detector (and their analysis like the cluster analysis described in 1.1.2) as I mentioned in the Introduction. It was developed at CERN for particle physics application.

The ROOT itself is a framework mainly written in C++ for big data processing, statistical analysis, visualisation, and storage [8]. For the purposes of this thesis I will use only the storage part of the ROOT.

Regardless it is written in C++ it allows the integration in other languages such as Python or R.

Chapter 2

Analysis

The analysis part is one of the cornerstones of software development in general, and the web application development in particular. It is a description of the final application behaviour. By the good analysis problems that can occur during the implementation and that would be difficult to remove after they occur can be avoided.

According to [9] and [10] the analysis contains a list of functional and non-functional requirements, use cases diagrams, activity diagrams, and the domain model. Except for the activity diagrams, which appeared simple enough to not present them, all above mentioned parts are presented.

2.1 Requirements analysis

Requirements analysis summarises the functional requirements and the non-functional requirements. The functional requirements describe what the application should be doing and the non-functional (general) requirements determine the needs for implementation, system efficiency and availability.

The requirements for the application are shown in the figure 2.1. Most of them were determined from the thesis assignment.

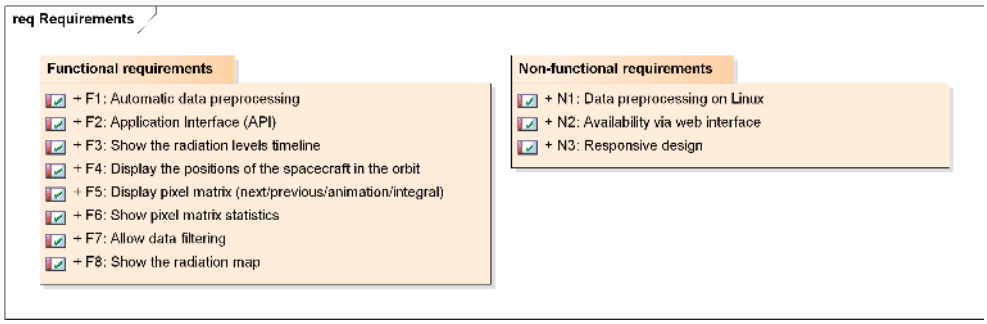

Figure 2.1: Overview chart of functional and non-functional requirements

2.1.1 Assignment fulfilment

Before going through all the requirements in detail, the assignment fulfilment is given by the functional requirements in the following:

1. Show the time dependence of the radiation levels.
2. Display the detector response in the form of a pixel matrix.
3. Show the positions of the spacecraft in orbit.
4. Implement an automatic data preprocessing
5. Create an efficient way for I/O operations to quickly scan huge datasets.
6. Define an application interface (API) for versatile query structure and integral data retrieval - this API will be used by the graphical user interface (GUI).

	F1 2.2.1	F2 2.2.2	F3 2.2.3	F4 2.2.4	F5 2.2.5	F6 2.2.6	F7 2.2.7	F8 2.2.8	fulfilled
1.			×						✓
2.					×	×			✓
3.				×					✓
4.	×								✓
5.	×								✓
6.		×							✓

Table 2.1: Table of the assignment fulfilment by the functional requirements

All the functional parts of the assignment are fulfilled by one or more functional requirements as you can see in the table 2.1. The functional requirements F7 and F8 do not fulfil any part of the assignment, because they were added above the assignment.

2.2 Functional requirements

The functional requirements were divided into two parts, the backend part, which describes all operations on the server and the frontend, which is handling the user interaction through a GUI (figure 2.2).

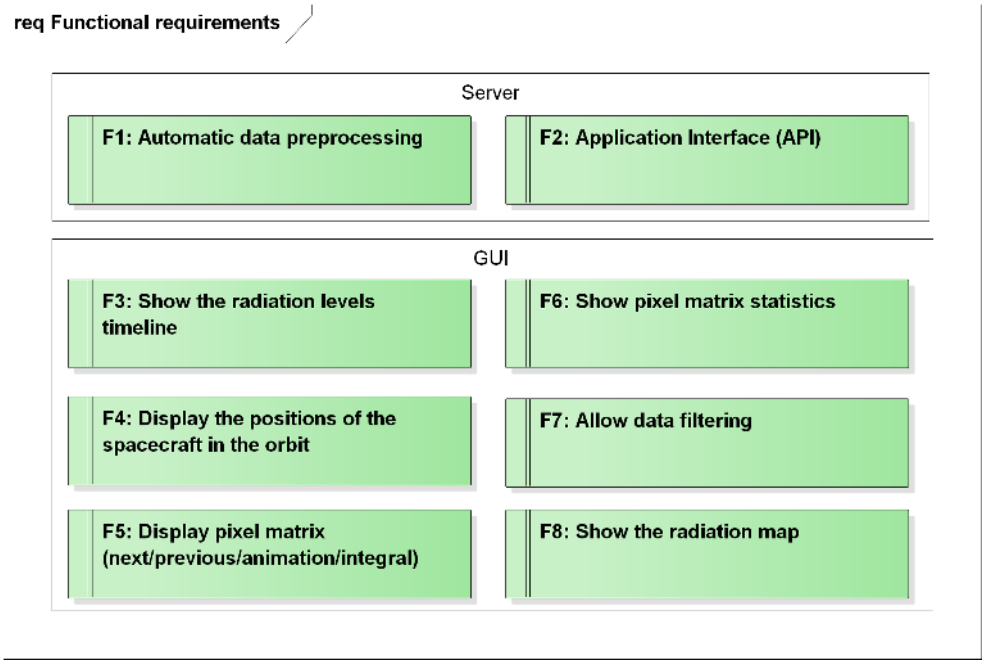

Figure 2.2: Functional requirements - the ”Server” part is performed on the server and ”GUI” part on a client (web browser)

2.2.1 F1: Automatic data preprocessing

The parser for the processing of the raw frame data is provided, but slight modifications in the code will be probably needed. The ROOT files are generated from this processing and their size is approximately 100-150MB (one file is generated per one day). It is needed to find an efficient way for I/O operations for these files to quickly scan them and retrieve the desired information to visualise it in the form of a web application. It means that the response time has to be fast - but the exact value of the response time to retrieve one frame has not been specified.

2.2.2 F2: Application Interface (API)

Application Interface (API) for versatile query structure and integral data retrieval will be defined. This API will be used by the GUI of the visualisation. It is preferred to use Representational State Transfer (REST).

2.2.3 F3: Show the radiation levels timeline

A timeline in the form of a graph visualising the radiation levels will be displayed in the GUI.

The center of the timeline will be the start time of the displayed frame and the user will be able to set the time range around it.

2.2.4 F4: Display the positions of the spacecraft in the orbit

The application will display the position of the Proba-V satellite in the orbit. The visualisation will be in 2D (on the map of the Earth) and/or 3D (the sphere is enough for the simulation of the Earth).

2.2.5 F5: Display the pixel matrix (next/previous/animation/integral)

The application shall visualise the pixel matrix of the Timepix detector - 256x256 pixels. There will be possibility to show next or previous matrix with the set acquisition time and also enter an animation mode that will periodically query the server for the next matrix. The animation mode should be able to recognise the end of the frames (there are no frames with the bigger start time) and continue from the start again.

The integral mode means that the matrix will be the sum of the frames specified by the start and end time.

2.2.6 F6: Show pixel matrix statistics

The number of tracks within the measured frame should be displayed as a table in the GUI. The statistics will be the number of tracks for each track type, the number of all tracks, total energy detected, and pixel matrix occupancy.

There will be a possibility to draw histograms of the cluster's statistics - e.g. volume, size, or maximum height. The cluster's statistics are already calculated by the cluster analysis and stored in the ROOT files.

2.2.7 F7: Allow data filtering

The user will be able to filter the frames by the following methods:

- **Start time**:
  The filtering by start time will be possible by selecting the desired time. The application will then display the nearest frame.

- **Acquisition time**:
  The user will be able to select the desired acquisition time(s) he wants to show. The Timepix detector that is used in the SATRAM has 3 acquisition times - 20s, 0.2s, and 0.002s (there are some frames with different acquisition times at the beginning of the measuring, but they will be not displayed).

- **Time range**:
  There should be a possibility to loop through defined time range in the application. It means that the user specify the start and end time for the frames and the application will show only the frames within this time range.

- **Satellite positions**:
  A world map in the GUI displaying the positions of the spacecraft will be shown. The user will be able to specify the desired area for the frames and the application will loop only through the frames within this area.

While the pixel matrix will be displayed the user will be able to apply filters on the cluster attributes. The filtration will be possible by following methods:

- **Cluster attributes**:
  The user will be able to filter each cluster by its attributes. The attributes are cluster volume, size, maximum height, linearity, elevation and azimuth, roundness, and length. All these attributes are calculated by the cluster analysis.

- **Track types**:
  The filtering by track types will be possible by selecting types like dots, small blobs, heavy blobs, heavy tracks, straight tracks, and/or curly tracks.

- **Mask**:
  The user will be able to create a mask for each pixel in the matrix. The masked pixel is therefore removed from all the frames and it is not calculated in the statistics.

- **Zoom**:
  The pixel matrix will allow the zoom functionality to show just pixels in some specified area. The shown area will be always a square to prevent pixels deformation.

2.2.8 F8: Show the radiation map

The application will be able to create a radiation map in the visualisation of the positions of the spacecraft in orbit. This means that it will calculate an average values of energies in the frames in some defined area and then draw the value (via a colormap) overlayed on the world map.

The user will specify a time range out of which the frames should be taken. The input is done by selecting the dates in a calendar input form.

2.3 Non-functional requirements

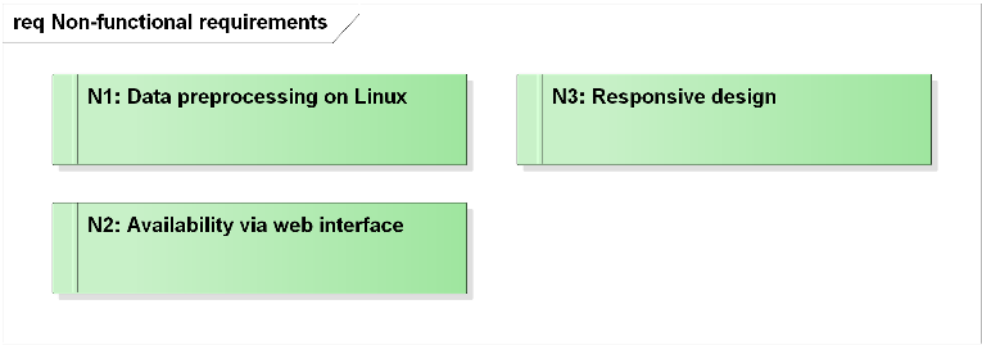

Figure 2.3: Non-functional requirements

2.3.1 N1: Data preprocessing on Linux

Data preprocessing (converting from raw frame data to ROOT files and preprocessing them) will be available on Linux/Unix based based system as an application ran from the command line.

2.3.2 N2: Availability via web interface

The final application will be available via web interface and browsers Google Chrome (version 54 or higher), Mozilla Firefox (version 47 and higher), and Internet Explorer (version 11 and higher - including Microsoft Edge).

2.3.3 N3: Responsive design

The application will have a responsive behaviour. It means that the application appearance will be different on different devices. This will not be achieved by javascript, but by media queries, flexible layout, and flexible images. By media queries, it is possible to define different appearance of the page for different devices/screen resolution [11].

2.4 Use cases

Use cases modelling is, according to [9], a detailed specification of the functional requirements. One functional requirement defined in the requirements analysis 2.2 is divided into one or more use cases.

Use cases can be used as the basis for creating manuals or for more accurate estimation of the labour input. It also serves as an assignment for the programmers that are creating the application.

The use cases model consists of following 3 main sections:

- **Actors**:
  It is a list of all users (human and nonhuman) that can use or affect the application.
- **Use case diagrams**:
  These diagrams display the connections between users and use cases.
- **List of use cases**:
  This list contains detail information to all use cases, scenarios (main and possible alternative) and conditions of the execution.

The list of use cases is included at the corresponding diagrams in the form of a text describing the use cases.

2.4.1 Actors

The application has 3 following types of actor:

- **Anonymous user**:
  The anonymous user is the user that can use the application GUI in all the possible ways.
- **Server**:
  The server is preprocessing the arriving data automatically once a day.
- **Other programmers**:
  The provided API could be used by other programmers in the future to retrieve desired data.

2.4.2 Use case diagrams

I divided the use cases diagrams into 3 parts - see figure 2.4: GUI visualisation is the part that will be seen by most users, data preprocessing will be done automatically, and the API could be used by other programmers in future (above that it is used by the GUI).

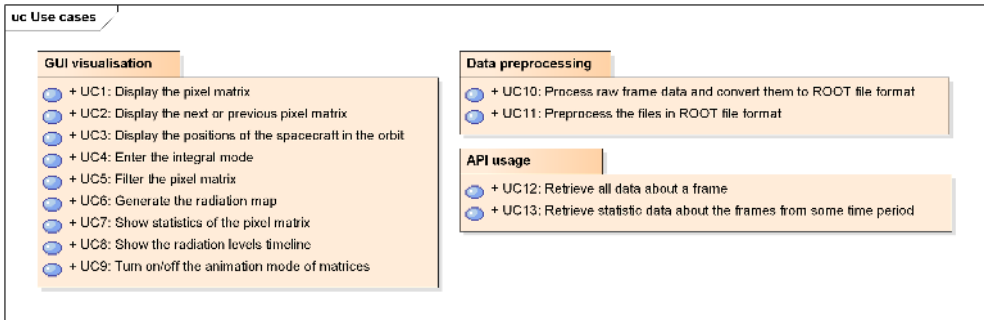

Figure 2.4: Overview of the use cases

2.4.2.1 GUI visualisation

This use case diagram displayed in figure 2.5 shows the part of the application with direct user interaction. The user has a GUI that shows the radiation levels timeline. The pixel matrix can be accessed via that (it can be done by input the precise desired time too). While the user has the pixel matrix displayed he can filter it by methods described in 2.2.5, show statistics, access previous or next matrix, turn on/off the animation mode of matrices, and enter the integral mode to display the desired number of frames as one image. He can also display the positions of the spacecraft in orbit or generate the entire radiation map from the positions of the spacecraft and the frames energies.

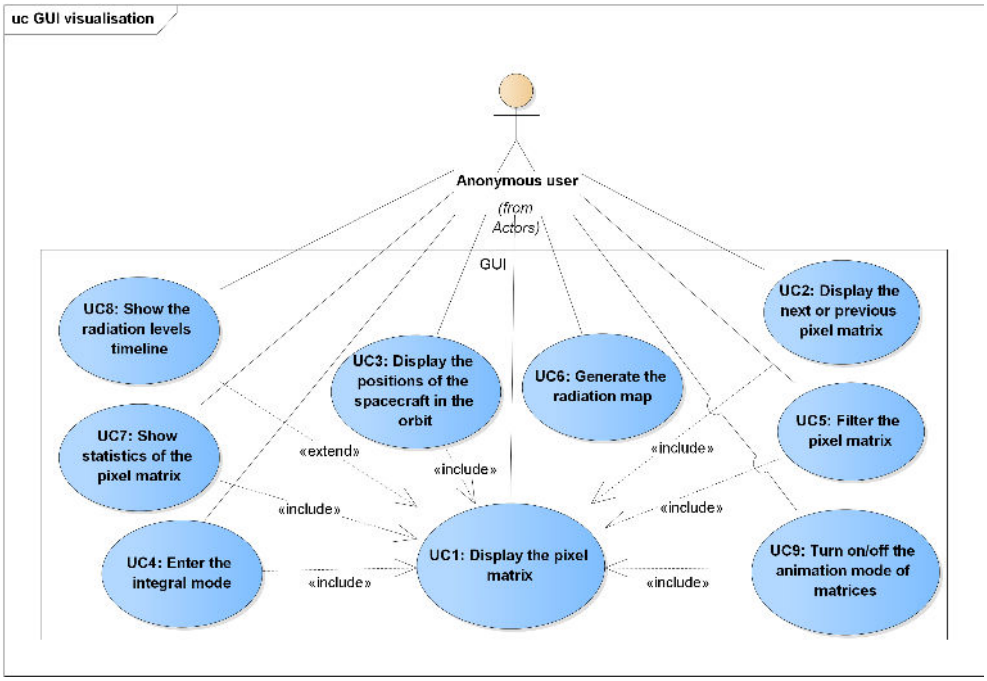

Figure 2.5: Use case of GUI visualisation

In the following, I describe the scenario in which the user displays the statistics about one pixel matrix.

Main scenario: display the pixel matrix statistics:

1. The scenario begins when the user want to display the statistics about some pixel matrix.
2. The GUI is showing the radiation levels timeline. The user click in the timeline and the application loads the pixel matrix from that time.
3. The user click in the menu on statistics and the GUI shows them.

Alternative scenario: display the pixel matrix statistics:

1. The scenario begins in the first step of main scenario.
2. The user fill the desired time from when he want to see the pixel matrix and send them. The GUI displays the nearest pixel matrix from the given time.
3. The user click in the menu on statistics and the GUI shows them.

The other scenarios in this use case diagram are straightforward in the way that the user just selects the functionality from the menu or visualisation - so I skip them.

2.4.2.2 Data processing

The data processing of the raw frame data is done automatically once a day (it is provided by the IEAP, but slight modifications can be needed). From this process the ROOT files are created. The application has to take these ROOT files as input and preprocesses them in some way to provide a quick response to requests from the GUI.

The conversion and the preprocessing can be done by the system administrator on demand, so the automatization of this process is not needed.

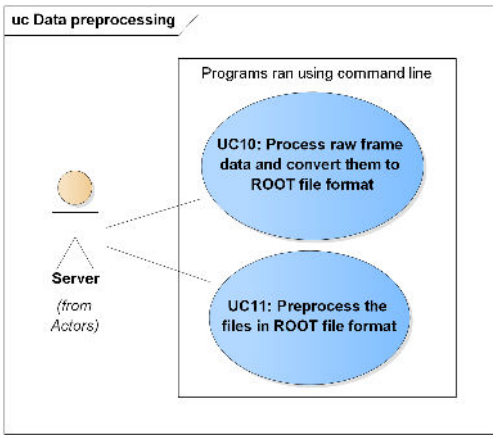

Figure 2.6: Use case of data processing

2.4.2.3 API usage

The API is primary used by the application GUI which displays the data to the user. But it is possible in the future that other programmers will use the API in the way of retrieving a pixel matrix or statistics from a time period (statistics like sum of the energies of all pixels in the matrix or pixel occupancy, not the entire frame's data) - see figure 2.7.

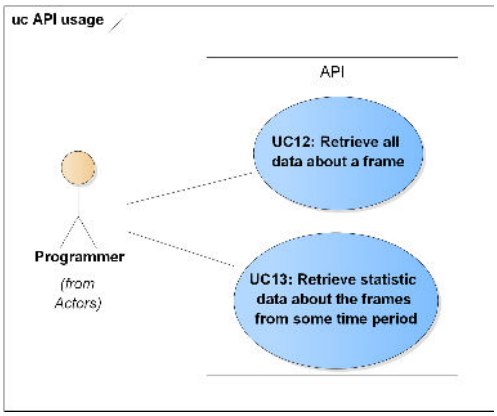

Figure 2.7: Use case of API usage

2.5 Check the fulfilment of all the functional requirements by the use cases

Table 2.2 shows the functional requirements fulfilment by all the use cases. If there would be a use case not fulfilling some functional requirement, it should be removed. On the other hand if there was a functional requirement not covered by any use case, a use case fulfilling this requirement would have to be added.

All the functional requirements are fulfilled by one or more use cases and no use case is unnecessary, as you can see in the last row of the table.

	F1	F2	F3	F4	F5	F6	F7	F8
UC1					×			
UC2					×			
UC3				×				
UC4					×			
UC5							×	
UC6								×
UC7						×		
UC8			×					
UC9					×			
UC10	×							
UC11	×							
UC12		×						
UC13		×						
fulfilled	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.2: Table of the functional requirements fulfilment by the use cases

2.6 Architecture scheme

Before the creation of the domain model it is essential to mention how the final application will work. A simple scheme showing the architecture of the application with considered technologies is illustrated in figure 2.8. The finally chosen technologies are discussed in the design chapter 3.

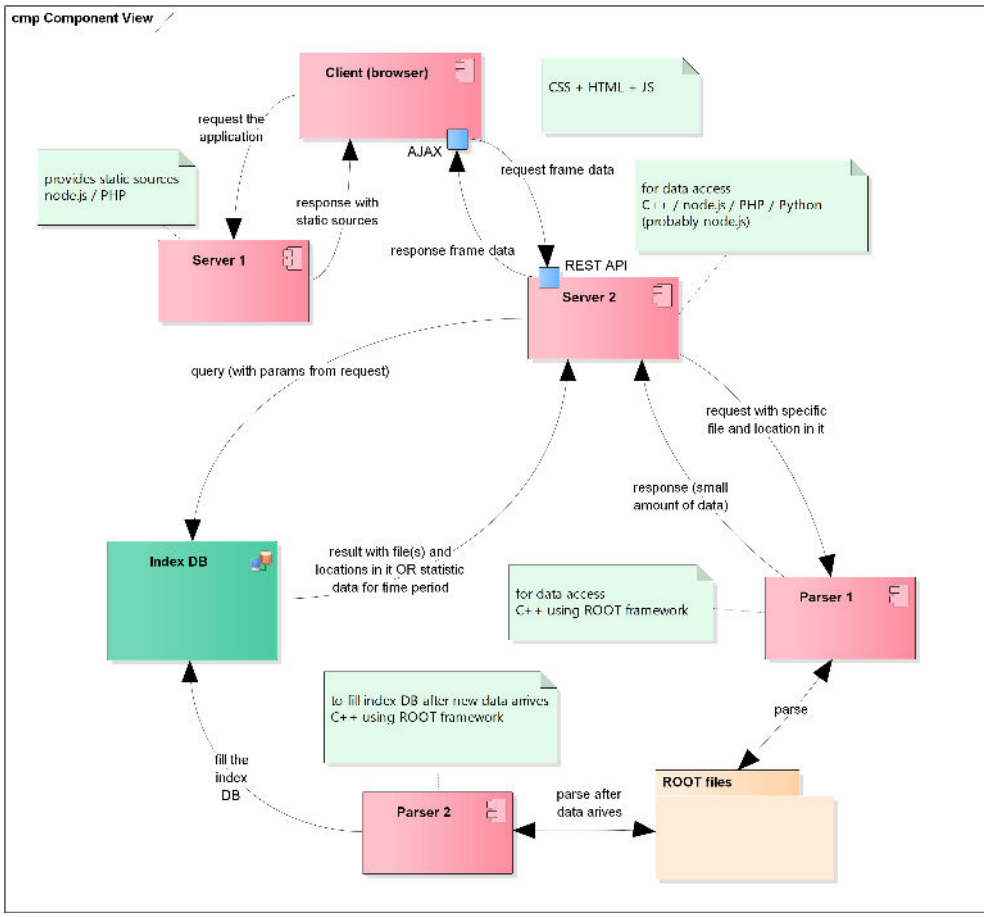

Figure 2.8: The component view

There will be 2 parsers for dealing with the binary ROOT files in the application. Parser 1 should know the exact positions of the desired information in the file, because it will be used to directly access the frame data in the application. Parser 2 will preprocess the new ROOT files and fill the index database (DB) mainly with the information to fast retrieve the data via the parser 1 and with the statistics data about the frame to draw for example the radiation timeline and map.

The client will request the static sources (entire GUI) from the server 1 - the HTML/CSS/JS files, pictures, and models for the 3D visualisation. The

requested GUI will then access server 2 for the data about the frames. Server 2 will be using the preprocessed data from the index DB to give it to the parser 1 to retrieve the desired information. It is possible that server 1 and 2 will be a single server in the end.

2.7 Domain model

There are two data sources in my application - the ROOT files (all data) and the index DB (positions in ROOT files, statistics data, and data needed for the radiation timeline and map generation). I will describe the index DB here, because it is the part that I created from scratch. The structure of the ROOT files is described in 3.3.2.2.

The domain model displayed in the figure 2.9 comes from the previous requirements analysis 2.1, where the requirements for the final application were defined.

The attributes needed for the whole realisation of the application should not be included in the analytical domain model. It should consist of the attributes needed for the understanding of the problematic. This is the reason why my domain model does not contain for example primary or foreign keys.

The goals of the domain model are according to [10]:

- describe the data
- describe relations between entities
- identify states of entities
- create the basis for design (the whole database model and class model)
- describe the meaning of terms
- capture attributes

Most of these goals are described in the figure 2.9. Only the identification of states of the entities is not described there because of the absence of entities with some state in my application.

2.7.1 Entities

There are two entities in the index database. The entity `rootfile` represents the actual root file on the disk and the entity `frame` represents the frames (pixel matrices) in the rootfile. There are data about clusters described in the frames, but I decided to not include them in the index database and access them directly via parser from the files to reduce the the data volume.

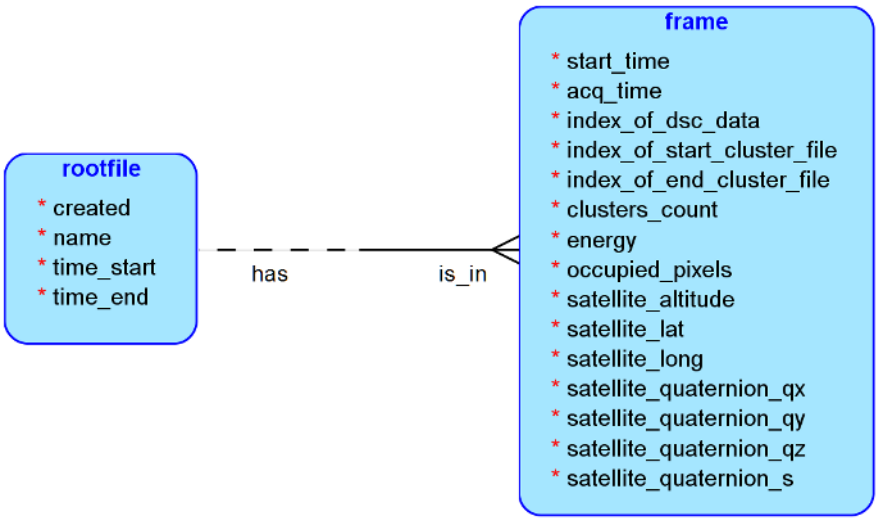

Figure 2.9: The domain model

2.7.1.1 Rootfile

The entity rootfile has 4 attributes:

- **created** - the date of creation to determine if the actual file was changed (recalculated) or not
- **name** - name of the file
- **time_start** and **time_end** - time range contained in the file

2.7.1.2 Frame

The entity frame has 15 attributes:

- **start_time** - the start time of the acquisition
- **acq_time** - the length of the acquisition
- **index_of_dsc_data** - the position in the `dscData` tree
- **index_of_start_cluster_file** and **index_of_end_cluster_file** - the positions in the `clusterFile` tree
- **clusters_count** - the number of clusters contained in the frame
- **energy** - the sum of all energies of pixels in the frame
- **occupied_pixels** - the number of pixels with some energy
- **satellite_altitude** - the satellite distance from the center of Earth

- **satellite_lat** and **satellite_long** - satellite position
- **satellite_quaternion_qx**, **satellite_quaternion_qy**, **satellite_quaternion_qz**, and **satellite_quaternion_s** - these numbers represents a part of the satellite rotation (more about it is described in 3.4.2.2)

Chapter 3

Design

This chapter is based on the previous analysis and consist of the relational model of the index DB, the GUI design, used technologies for implementation, and implemented algorithms and methods.

3.1 Relational model

The relational model comes directly from the domain model described in 2.7. It contains the data types, primary keys, foreign keys, and indexes used in the final database.

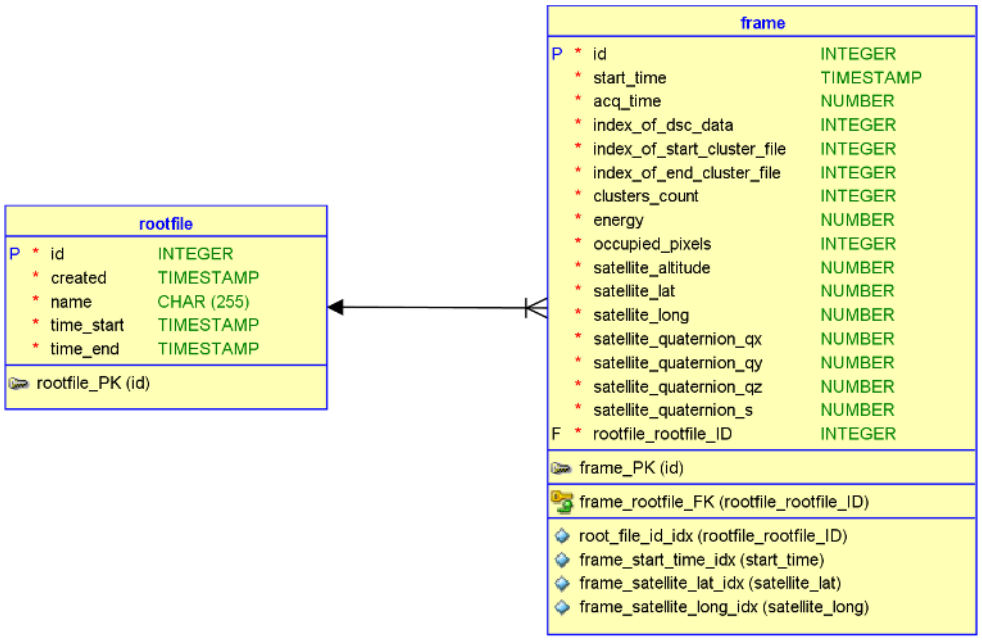

Figure 3.1: The relational model

Both entities have the primary key `id` as you can see in the figure 3.1. The entity frame contains the foreign key to the rootfile and 4 indexes. The indexes are used by SQL algorithms to provide a quick response to the query that contains the attributes with indexes [12].

Most of the queries will be probably using the start time of the frame to access the right frame, so there is an index above it. There will be a possibility to filter the frames by the positions of the spacecraft in orbit in the GUI, so an index for the latitude and longitude was created.

It is worth mentioning that all the timestamps are without any timezone. There will be a clear identification that all the times are in the Coordinated Universal Time (UTC) format in the GUI. It is implemented this way to prevent any problems during a communication with a person in different timezone.

3.2 Graphical user interface design

I used a technique called wireframing to design the graphical user interface. The wireframe is a simple representation of all elements of a web page and how they fit together [13]. It is mainly used to clarify the positions of all elements, not their design. This is the reason why wireframes does not have pictures or colours.

There are many programs to create the wireframes, but probably the best is to use a pencil and paper [14]. The reason for this is that the pencil can be erased from the paper, it looks creative, and it is fast for a quick sketch of the page. It works in practice, but ,for a thesis like this, it does not look good. This is the reason why I created the wireframe on paper first and then used a software to redraw it into a more representative look. There are many software options for wireframe creation e.g. Balsamiq[4], Google draw[5], or Pencil[6]. In this thesis, Pencil was used since it is open source and has all necessary functionalities.

A single page application layout was chosen - the browser loads the page just once and all changes of the GUI are done by Javascript. Besides the switching of the contents in the menu bar, not many changes in the final application layout are needed, so it was sufficient to create only one wireframe. This wireframe is depicted in figure 3.2.

There were a lot of changes in the page layout during the implementation mainly due to resizing of the elements, e.g. the physicists working with the pixels detectors are used to have the pixel matrix bigger than 256x256 pixels. Or the timeline was across the whole page width and when I put there a graph

[4]https://balsamiq.com/
[5]https://docs.google.com/drawings
[6]http://pencil.evolus.vn/

it was clear to me that it is too wide for the presented data, so I reduced the width to 50% of the page. The presented wireframe is the final version.

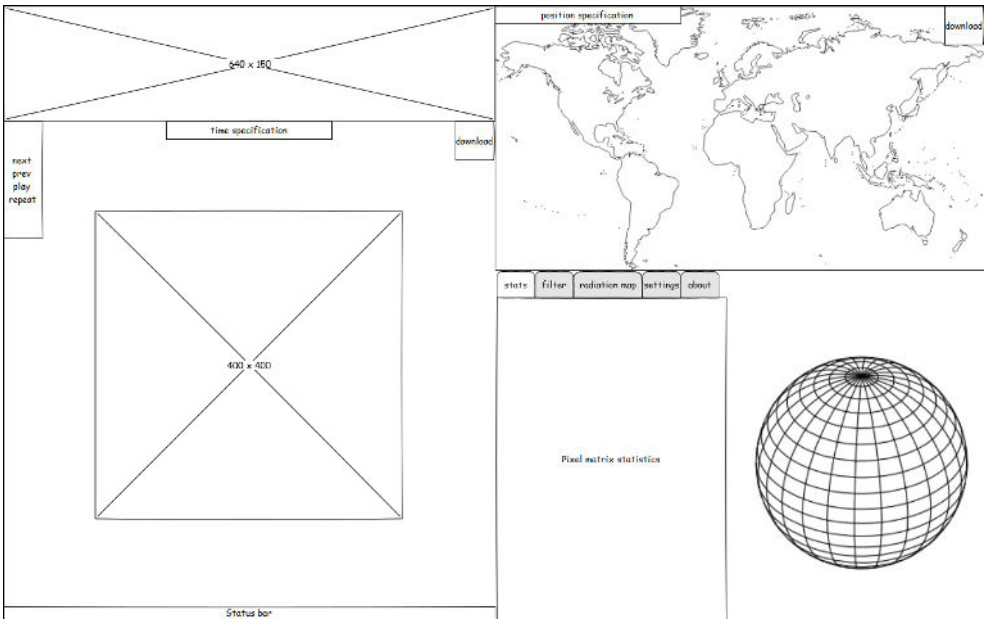

Figure 3.2: The main wireframe

The main part of the application is on the left side. There is the pixel matrix in the area. You can see the controls to access the next, previous, animation mode, and setting to repeat the frames in a loop on the top left side in the area. There is a possibility to download the matrix in the top right corner. The information about the start time and acquisition time are centred on the top. You can see the status bar here to display if there are some requests to the server and how long their execution took.

There is the radiation timeline above the frame area. It is a line graph of the radiation around the displayed frame.

A big area of the screen is taken by the spacecraft position visualisation. A 2D world map is placed in the upper right part with a button allowing a download as a png file. It also shows the result of the radiation map generation. The 3D visualisation is positioned in the bottom right corner.

Most of the controls are on the left side of the 3D visualisation. You can see the pixel matrix statistics, filter possibilities, radiation map generation, settings, and the about section here. The final menu items are represented by icons with labels because of the space and more reliable behaviour regarding the page responsibility.

3.3 Used technologies

There are many technologies to be used in the web development. Since the application is client-server, I divided this section into two parts: client and server. I briefly introduce every technology I used and I explain why I have decided to use it.

3.3.1 Client part

The client part is the part interpreted by the browser. The HyperText Markup Language (HTML) is used for data representation, Cascading Style Sheets (CSS) for better visualisation of the data represented by HTML, and Javascript (JS) for scripting and most interactions with the user.

I used some technologies/libraries above this 3 cornerstones of the web application. I describe them in the following sections.

3.3.1.1 HTML

The first thing in the HTML element should be the Document Type Definition (DTD) often called a doctype. The doctype is used for two following things according to [15]:

- Web browsers know which rendering mode to use (it defines the version of (X)HTML).
- Validators use it to determine which rules they should check the document against.

The decision which version of (X)HTML I should use was quite easy for my application, because I knew that I needed to use the `<canvas>` element for most parts of my application (timeline, pixel matrix, and 3D visualisation). The element allows the programmer to draw content inside using a javascript API. The only version of (X)HTML that is supporting this element and I used it for the implementation is HTML5[7].

3.3.1.2 Sass

Syntactically Awesome StyleSheets (Sass)[8] is a CSS preprocessor. It allows the programmer for example to define variables or inheritance of selectors.

The preprocessor take the Sass file(s) and convert them into the final CSS file(s). Upon execution of the conversion, the programmer can define options such as the minification or the CSS version. It is mainly used to make the programmer's work easier.

[7] https://www.w3schools.com/html/html5_intro.asp
[8] http://sass-lang.com/

You can see Sass in action in the following example from the final application that shows the import of other files, the inheritance of selectors and how to use variables:

```
@import "variables";
div.statistics {
  div.worldMap {
    border-left: 2px solid $linesColor;
    dl.mapStatisticsArea {
      font-size: 70%;
      background-color: $defaultBGColor;
    }
  }
}
```

3.3.1.3 Bootstrap

Bootstrap[9] is one of the most popular CSS and JS framework. It uses Normalize.css[10] to improve the cross-browser rendering - it unites paddings, margins, headings sizes, and many more attributes that can be different in various browsers. The unification is one of the features that influenced me to use this framework.

The grid system is an important part of Bootstrap too. The programmer can define the number of columns that some area takes (whole page is 12) for different types of devices - `col-lg-N` for large devices and desktops, `col-md-N` for medium devices and desktops, `col-sm-N` for small devices and tablets, and `col-md-N` for extra small devices and phones where N is the number of columns. The switching of these states is achieved by the detection of the page resolution, so it influenced even the behaviour when you make the browser window smaller.

Bootstrap also influences the basic look of the forms, buttons, checkboxes, and all the basic HTML elements on the web page. It is achieved by adding some class to the element. I use this to unify the look of the application.

Last but not least I should mention here are icons. Bootstrap provides 263 icons for various occasions. I used many of the icons in the final application to save some space (for example menu items are displayed as icons with labels).

3.3.1.4 TypeScript

TypeScript[11] is a Javascript preprocessor. The TypeScript files are compiled into the Javascript files in the command line. The programmer can specify the version of ECMAScript for the compilation.

[9] http://getbootstrap.com
[10] http://necolas.github.io/normalize.css/
[11] https://www.typescriptlang.org/

The name can suggest that the TypeScript would have something to do with types. And it is not wrong. The programmer can specify the type of his variables (numbers, strings, objects,...) and the compilation will look at it and throw some error for example when you try to add a number and string.

Beside the types, TypeScript supports creating classes, interfaces, and inheritance. Someone could say that you can define a class in the pure Javascript too. He would not be wrong, but the inheritance in the pure Javascript is not simple and can behave unpredictable when the programmer forgets to define all necessary elements.

To sum up, TypeScript allows the programmer to write clean and efficient code that is reusable. It allows to create the modern design patterns pretty easy and it still provides all the advantages of Javascript.

3.3.1.5 JQuery

JQuery[12] is a Javascript library. It handles the HTML elements manipulation, event registration, animations, and Ajax (Asynchronous JavaScript and XML) much simpler than using the pure Javascript API for this purpose.

I use JQuery mainly for identification of a set of elements easy by CSS selectors and iteration through them via JQuery `$.each()` method. Also Ajax request are easy to perform using for example `$.ajax()` method.

I decided to use JQuery just to simplify working with the HTML elements and performing Ajax requests.

3.3.1.6 Google Charts

There are many javasript libraries for drawing graphs on the web page in the web environment - e.g. ChartJS[13], Chartist[14], ZingChart[15], or Google Charts[16].

I needed to draw a chart in the form of the radiation timeline (line chart) and some histograms for the pixel matrix statistics. Most of the mentioned libraries have this basic drawing abilities. In addition, I needed some interaction with the radiation timeline (click on the specified data). I knew that Google Charts API provides this functionality in a simple way, so I decided to use this library. The fact that I had a very positive experience with the performance of Google Charts API also contributed to the decision.

[12] https://jquery.com/
[13] http://www.chartjs.org/
[14] http://gionkunz.github.io/chartist-js/
[15] https://www.zingchart.com/
[16] https://developers.google.com/chart/

3.3.1.7 Three.js

WebGL is a web standard for 3D graphics based on OpenGL, exposed to ECMAScript through the HTML5 `<canvas>` element. WebGL program consists of a control code in JavaScript and a shader code executed on a Graphics Processing Unit (GPU).

Because the pure WebGL is pretty low-level I decided to use a javascript library for it. There are many libraries for this purpose in the web environment. Probably most known is Babylon[17] that supports scene graphs with lights, cameras, materials and meshes, collisions engine, physics engine, audio engine, and optimisation engine at the core. It is the engine for most of the 3D games based on WebGL. Babylon is pretty complex and has many features that were not needed, so I decided to use some more simple library - Three[18].

Three is a little bit older than Babylon and supports less features, but it includes the features I needed, which were the loading of meshes with materials and putting some lights in the scene for the whole 3D visualisation in the final application. The engine did not have to be as complex as Babylon and therefore Three was chosen.

3.3.2 Server part

The server part is the part running on the server. It provides suitable responses for user's requests. Unlike the client part, the server part can be written in many totally different languages. I describe the languages and technologies I used to create the final application in the following sections.

3.3.2.1 Server

I decided to use only one server instead of possible two servers displayed in the figure 2.8. The choice of the final language was very important. I describe the considered possibilities in detail in following sections. I considered C++, PHP and Node.js.

3.3.2.1.1 C++ I knew that I probably use C++ for the ROOT files access, so the whole server part in C++ programming language was the first what came to my mind.

Facebook provides an open source library called Proxygen[19] for exactly this use. The Proxygen's goal is to provide a simple, efficient, and modern C++ HTTP (Hypertext Transfer Protocol - the base protocol for the communication in the World Wide Web) library.

[17] https://www.babylonjs.com/
[18] https://threejs.org/
[19] https://github.com/facebook/proxygen

C++ is known for its good performance and the efficient memory management managed by the programmer, so it would be probably the best programming language to use regarding the code efficiency. However, I decided not to use C++ because the time requirement estimated for this way was to high for the benefit it can potentially bring.

3.3.2.1.2 PHP There are some ways to connect C++ code to the PHP application for example php-cpp[20] extension, but I assumed that the ways are more difficult and problematic than in Node.js integration. This is the main reason, why I decided to write the final server in Node.js. I also consulted the decision with programmers in IEAP (the institute that will be using the final application) and they preferred server in C++ or Node.js because of the good knowledge of these languages.

3.3.2.1.3 Node.js Node.js[21] is the language I decided to use to implement the server part in the application. It is a JavaScript runtime built on Chrome's V8 JavaScript engine that is written in C++, so it has a potential for a good performance and easy interaction with C++ code used for the ROOT files access.

Node.js is an asynchronous event driven runtime and it is lightweight and efficient. It is designed to build scalable network applications and it can handle many connections concurrently. The programmer can write C++ addon directly through V8 engine - see 3.3.2.3.

3.3.2.2 ROOT files access

ROOT files are binary files, so it can be accessed directly with the knowledge of their structure. Fortunately, CERN provides a framework for it conveniently called ROOT framework[22]. It is a modular scientific software framework that provides big data processing, statistical analysis, visualisation, and storage. I used the visualisation part for a quick look at the files - it draws histograms of all stored variables within the file and shows its structure, but in the final application only the storage part is used.

The framework is written mainly in C++, so the integration with it is quite easy. It can be integrated with other languages like Python or R, but C++ is the best way for me because of its good integration with Node.js.

It is worth mentioning that the ROOT framework was born at CERN, at the heart of the research on high-energy physics and physicists I have been working with are using this framework every day to analyse the data or perform simulations.

[20] http://www.php-cpp.com/
[21] https://nodejs.org/en/
[22] https://root.cern.ch/

3.3.2.3 Integration of the ROOT framework and server

The interaction of the server in Node.js and the root file access in C++ language is the last piece of the puzzle. There are a few ways how to do it according to [16]:

- **Running an executable in the command line**:
  Node.js supports spawning of a child process[23]. It provides both synchronous and asynchronous version of running an executable file in the command line. It is the easiest way from all, but it has some serious disadvantages: poor performance, scaling, and the executable is run every time, so there is no way to implement some caching mechanism and the file has to be opened and closed every request. It was a big problem because I was dealing with file sizes in hundreds of megabytes. This was the reason I immediately rejected this way.

- **Creating a shared library (dll)**:
  This option is mainly used when you have some dll/lib. The programmer can call a routine through some interface (for example Foreign Function Interface[24]). It is closer to the full integration, but there are problems with type conversions and mainly blocking when calling the C++ routine. Blocking the entire process in a web server is not a good practise, because when there are many users using the application each of them has to wait until all the users before him get their data. It is even multiplied by the fact that I was dealing with big file sizes, so I rejected this way too.

- **Node.js addon**:
  Creating Node.js addon is a complete integration of the C++ code into the Node.js application. It is more challenging than the two previous approaches, but the reward is flexibility and performance. I can let the ROOT file open for more than one request and the addon can be implemented asynchronously, so it does not block the entire application. I assumed that this is the hardest but the best way to implement the integration, so I used this approach.

3.4 Used algorithms and methods

There are some methods in the final application that require a little explaining theory for the complete understanding. I explain the theory here in the following sections and their implementation in the Realisation chapter 4.

[23] https://nodejs.org/api/child_process.html
[24] https://github.com/node-ffi/node-ffi

3.4.1 Mercator projection

There is a part of the application where I need to convert the satellite position from `latitude` and `longitude` to the 2D coordinates `x` and `y`. Many projections serves this purpose. Because there were no requirements regarding the position accuracy (the exact position is shown in the 3D visualisation) I decided to use a projection that is used by Google, the Mercator projection.

The Mercator projection deserves more than just one page, but it is not the main part of the final application, so I describe just a bare minimum needed for the implementation. For more information about this topic the interested reader may refer to [17].

The Mercator projection is a normal cylindrical map projection. It means that the projection is defined on a cylinder which is tangential to the sphere on the equator (figure 3.3).

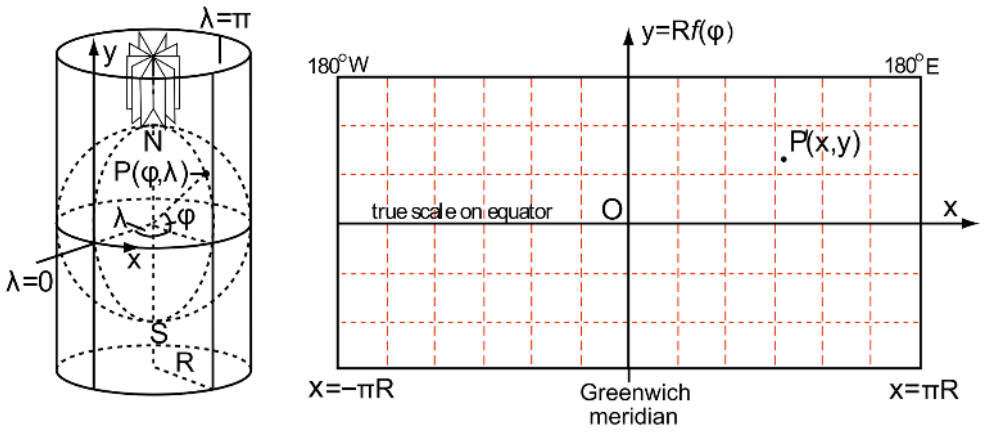

Figure 3.3: The normal cylindrical projection; source: [17]

The formula for the conversion of the `latitude` and `longitude` to `x` and `y` is following:

$$x = (lon + 180)\frac{w}{360}$$

$$y = \frac{h}{2} - \frac{h \cdot ln(tg(\frac{\pi}{4} + \frac{lat\frac{\pi}{180}}{2}))}{2\pi}$$

The variables `lon` and `lat` correspond to the `latitude` and `longitude`. The variable `w` is the underlying map width and `h` is the underlying map height.

It can be quite confusing for the first time, but if you look closer it is quite easy. The `x` is calculated from the `longitude` directly without any distortion and `y` is calculated from the `latitude` with the use of the natural logarithm, so the area around the Equator will have more satellite positions than areas around poles.

3.4.2 Satellite rotation

The satellite position is defined by the `longitude`, `latitude`, and `altitude`. The satellite rotation is defined a little bit more complicated, so I describe it in the following sections. The source of all the information was [18].

3.4.2.1 Frames

A few coordinate systems (frames) are defined for every satellite. The data are always defined in one of the frames, usually in a frame related to the satellite. I want to visualise the satellite in a frame relative to the Earth in the final visualisation to display the satellite position in the orbit. The frames defined in Proba-V data are briefly described in the following sections.

3.4.2.1.1 Body Fixed Frame (BOF frame) This frame is fixed on the satellite with its origin in the center of the satellite's mass. You can see the illustration of the frame on the figure 3.4.

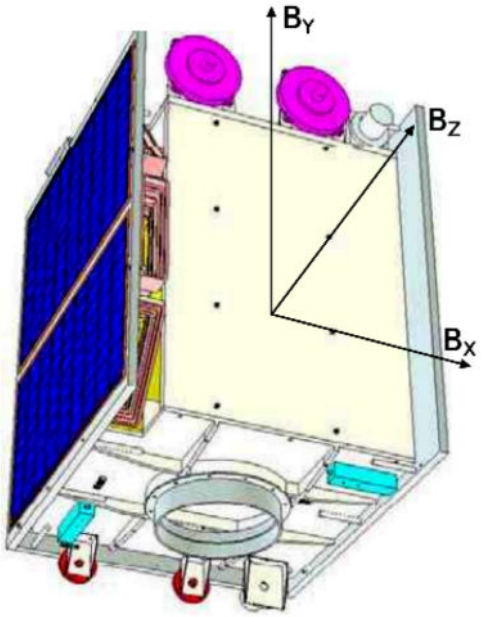

Figure 3.4: The illustration of the satellite's BOF frame; source: [18]

The axes of the frame are labelled B_x, B_y, and B_z. The z-axis should always point to the Earth because of some Proba-V imager observing the Earth - this side of the satellite is called Nadir Panel. The x-axis is parallel to the velocity vector of the satellite - the side is called the Velocity Panel. The y-axis is perpendicular to the z-axis and x-axis in the direction according to the right-hand rule.

All names and abbreviations of the Proba-V's panels are defined in the following table 3.1. The SATRAM payload is placed on the Bottom Board.

Panel	Abbreviation	Direction
Nadir Panel	NP	z
Zenith Panel	ZP	-z
Velocity Panel	VP	x
Anti-Velocity Panel	AP	-x
Top Panel	TP	y
Bottom Board	BB	-y

Table 3.1: Panels of the Proba-V satellite; source: [19]

3.4.2.1.2 Earth Centred Fixed Frame (ECF frame) This frame is fixed on the Earth with its origin in the center of the Earth's mass and it is rotating with the Earth. The axes of the frame are labelled ECF_x, ECF_y, and ECF_z. The z-axis is parallel with the rotating axis of the Earth. The x-axis points to the intersection of the equator and the Greenwich meridian. The y-axis is like in the BOF frame perpendicular to the other two in the direction according to the right-hand rule.

3.4.2.1.3 Earth Centred Inertial frame (ECI frame) This frame is fixed on the Earth with its origin in the center of the Earth's mass, but unlike ECF frame it is not rotating with the Earth (this system is International Celestial Reference System). It is also called CEL frame in the case of Proba-V. The axes of the frame are labelled ECI_x, ECI_y, and ECI_z. The z-axis is parallel with the rotating axis of the Earth. The x-axis points to the vernal equinox. The equinox is the moment in which the plane of Earth's equator passes through the center of the Sun. It occurs twice a year - in the March and September. The vernal equinox is the one in the March. The y-axis is perpendicular to the z-axis and x-axis in the direction according to the right-hand rule.

This frame was created to have an inertial coordinate system nearly fixed in space. There are defined certain points in time to be taken as a reference, because it is not perfectly fixed in space. For most satellites including Proba-V the J2000.0 epoch is used. This epoch refer to 1.1. 2000 12:00 and to transform the frame to the ECF frame the rotation of the Earth since that date has to be considered.

3.4.2.2 Quaternions

Now when we know the theory about the satellite frames we can describe the needed transitions between them. The quaternions are used for the transition from BOF to the ECI frame. They are a mathematical construct based on

the Euler theorem that states that every transformation between 2 coordinate systems can be described as a rotation around an axis by an angle.

I describe the theory about quaternions needed to implement the transition from BOF to the ECI frame in this section. It is worth mentioning that quaternions are widely used to describe the attitude of satellites.

A quaternion is a vector of 4 components that consists of a scalar part s and a vector part $\vec{q}$. The scalar part is usually defined as the first element, but for the Proba-V it is defined as the last element:

$$q = \begin{bmatrix} \vec{q} \\ s \end{bmatrix} = \begin{bmatrix} \vec{q}_x \\ \vec{q}_y \\ \vec{q}_z \\ s \end{bmatrix} = \begin{bmatrix} \vec{k} \cdot \sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{bmatrix}$$

where θ is the rotation angle and $\vec{k}$ is the rotation axis.

A quaternion has to be normalised to use it for the attitude representation. To normalise it just divide it by its norm:

$$||q|| = \frac{q}{|q|}$$

where

$$|q| = \sqrt{q_x^2 + q_y^2 + q_z^2 + s^2}$$

Someone should want the rotation by an opposite angle which is defined by an inverse quaternion that is the normalisation of its conjugate:

$$q^{-1} = \frac{q^*}{|q|}$$

where

$$q^* = \begin{bmatrix} -\vec{q}_x \\ -\vec{q}_y \\ -\vec{q}_z \\ s \end{bmatrix}$$

The product of quaternions q_1 and q_2 is:

$$q_1 \otimes q_2 = \begin{bmatrix} s_1\vec{q}_1 + s_2\vec{q}_2 + \vec{q}_1 \times \vec{q}_2 \\ s_1 s_2 - \vec{q}_1 \circ \vec{q}_2 \end{bmatrix}$$

where $\times$ is the cross product and $\circ$ is the dot product.

3.4.2.3 Final rotation

The final transition which has to be done to properly visualise the satellite attitude is the transition from ECI to ECF frame. The z-axis of both frames are the same, so it is a simple rotation around the z-axis in the x-y-plane. The rotation angle is determined from the fact that the ECI frame the J2000.0 epoch is used for Proba-V. The angle is called Earth Rotation Angle (ERA) and it is calculated with the use of Julian UT1 day - UT1 is the Universal Time which is currently used for the International Celestial Reference System. The ERA angle is defined:

$$ERA = 2\pi(0.7790572732640 + 1.00273781191135448T_U)rad$$

where T_U is the number of days past since the 1.1. 2000 12:00.

3.4.2.4 The whole rotation

Now we know the theory needed to perform the whole rotation. I describe the rotation step by step in this section to clarify the process. The rotation consists of 2 steps:

1. transition from **BOF to ECI** frame - with the help of quaternions
2. transition from **ECI to ECF** frame - rotation around z-axis by the ERA angle

I describe this 2 steps in the following sections, where:

- $\vec{v}_{BOF} = (x_{BOF}, y_{BOF}, z_{BOF})^T$ is a vector in the BOF frame
- $\vec{v}_{BOF} = (x_{BOF}, y_{BOF}, z_{BOF})^T$ is a vector in the BOF frame
- $\vec{v}_{ECF} = (x_{ECF}, y_{ECF}, z_{ECF})^T$ is a vector in the ECF frame

3.4.2.4.1 BOF to ECI To perform this rotation the following equation has to be solved:

$$\begin{bmatrix} \vec{v}_{ECI} \\ 0 \end{bmatrix} = q \otimes \begin{bmatrix} \vec{v}_{BOF} \\ 0 \end{bmatrix} \otimes q^{-1}$$

where q is a quaternion and an input parameter (provided by Proba-V) and $\otimes$ is the product of quaternions. From this equation the rotation matrix can be defined:

$$M_1 = \begin{pmatrix} s^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_xq_y - sq_z) & 2(q_xq_z + sq_y) \\ 2(q_xq_y + sq_z) & s^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_yq_z - sq_x) \\ 2(q_xq_z - sq_y) & 2(q_yq_z + sq_x) & s^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix}$$

Then to get the vector in ECI frame the simple matrix multiplication has to be calculated:

$$\vec{v}_{ECI} = M_1 \cdot \vec{v}_{BOF}$$

3.4.2.4.2 ECI to ECF This rotation is a simple rotation around z-axis by the ERA angle, so the following rotation matrix can be defined:

$$M_2 = \begin{pmatrix} \cos(ERA) & \sin(ERA) & 0 \\ -\sin(ERA) & \cos(ERA) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where ERA is the angle defined in 3.4.2.3. Then to get the vector in ECI frame the simple matrix multiplication has to be calculated:

$$\vec{v}_{ECF} = M_2 \cdot \vec{v}_{ECI}$$

Now we have a vector in the ECF frame - the frame needed for the final visualisation of the satellite attitude relative to the Earth.

3.5 Dose rates

There is a possibility to show "dose rates" in the radiation timeline and radiation map. Absorbed dose represents the mean energy imparted to matter per unit mass by ionizing radiation [20]. Its unit is Gray (Gy) and the general formula for its computation is:

$$\overline{D}_T = \frac{\int_T D(x,y,z)\rho(x,y,z)dV}{\int_T \rho(x,y,z)dV}$$

where:

- $\overline{D}_T$ = the averaged absorbed dose of the entire item T
- T = the item
- $D(x,y,z)$ = the absorbed dose as a function of location
- $\rho(x,y,z)$ = the density as a function of location
- V = the volume.

After putting the variables from the Timepix detector into the formula we get the following:

$$\overline{D}_T = \frac{E \cdot 1.6 \cdot 10^{-13}}{1.4^2 \cdot 0.03 \cdot 2.3}$$

where E is the sum of the energy stored within the frame.

Chapter 4

Realisation

This chapter presents the realisation of main parts of the application - data preprocessor, server part, and client part. Used technologies were described in 3.3, so I focus just on the implementation of the key elements here.

I present an Unified Modelling Language (UML) class diagram with the essential parts where it is suitable. A UML class diagram is a type of diagram that describes the structure of a system by showing its classes, their attributes, operations, and relationships [21].

4.1 Data preprocessor

The data preprocessor takes a ROOT file and creates the database records in the index DB. It just preprocess the data to be used by the server part of the application described in 4.2.

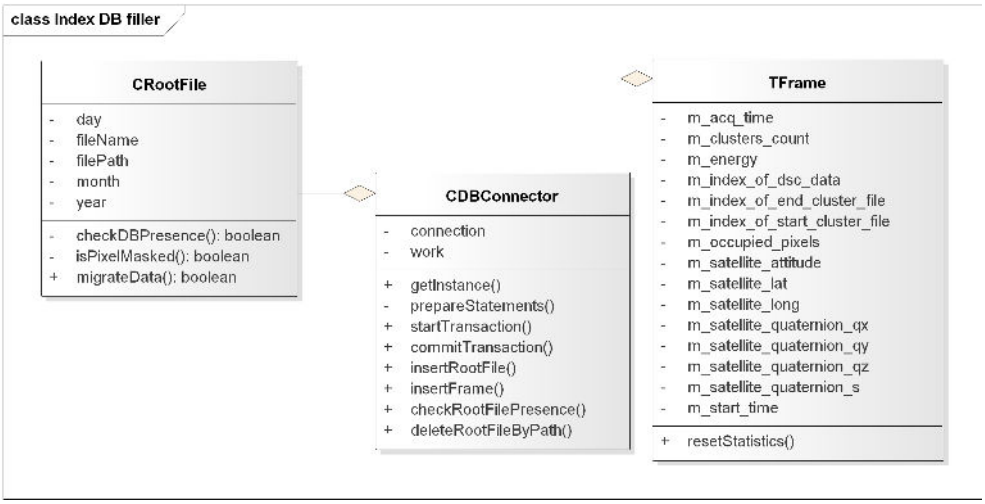

Figure 4.1: Class diagram of the data preprocessor

The class diagram of the data preprocessor can be seen in the figure 4.1. The classes are described in the following sections.

4.1.1 CRootFile

The class `CRootFile` represents a root file. It has attributes that represents data about the file (`day`, `month`, `year`, `fileName`, and `filePath`).

The private method `isPixelMasked` is used to find out if a specified pixel is masked or not. Some pixels have to be masked due to a firmware problem in the detector. For each ROOT file a vector of 65536 values (0/1) is stored in the `calibData` tree. This values represents the 256x256 values of the pixel matrix and states if the pixel is masked or not.

Another private method `checkDBPresence` checks if the ROOT file is in the index DB. It uses the `CDBConnector` for it.

The most important method is `migrateData` which performs the insertion of data to the index DB using `CDBConnector`. It loads all needed data about the file and inserts them to the index DB. Then it loops over all the frames, calculates all the statistics and stores them in the `TFrame` structure which is then inserted into the DB. The commit is performed after all insertions, retarding the ROOT file to increase the performance.

4.1.2 TFrame

`TFrame` is just a structure used by `CRootFile` to save statistics of the frame and then store them into the DB.

4.1.3 CDBConnector

`CDBConnector` is a class providing the connection with the DB. It is implemented as a singleton design pattern which ensures that it has one instance and provides a global point of access to it [22]. For the connection, the `libpqxx`[25] library is used.

The private function `prepareStatements` creates the so called "prepared statements". Prepared statements are SQL queries that you define once and then invoke multiple times, typically with varying parameters. The most used statements are for the rootfile and frame insertion. They are used by the functions `insertFrame` and `insertRootFile`.

Another important function is `checkRootFilePresence`, which checks if the ROOT file is already in the DB and, if the record is old (file was updated e. g. recalculated by cluster analysis), the function removes it.

[25] http://pqxx.org/development/libpqxx/

4.2 Server

The server part is written in Node.js. The code is divided into a few modules: server, db-connector, root-files-access (using tpx-frame-accessor - C++ addon), radiation-timeline, and radiation-map. I describe each module in the following sections.

4.2.1 Server

The file server.js contains the code which serves the incoming requests. It provides the Application Programming Interface (API) that consists of the following entry points (all only for the GET method):

- **/api/frame**

  This is the most important entry point. It gives information about the frame(s) and accepts the following parameters:

  - **time** - timestamp of the start time of the first frame
  - **acqTime** - acquisition time(s) of the desired frame(s)
  - **numberOfFrames** - number of desired frames (time is for the first frame)
  - **from** (optional) and **to** (optional) - for time filtering
  - **minLat** (optional), **maxLat** (optional), **minLong** (optional), and **maxLong** (optional) - for position filtering

- **/api/radiationMap**

  This is used to get information to generate the radiation map and accepts following parameters:

  - **acqTime** - acquisition time(s) of the frames
  - **from** and **to** - time range of the frames

- **/api/radiationTimeline**

  This is used to get information to draw the radiation timeline and accepts following parameters:

  - **acqTime** - acquisition time(s) of the frames
  - **from** and **to** - time range of the frames

This module also checks the parameters for the wrong values and responses accordingly to the bad requests, and uses the helmet[26] module to provide better security. The other functionality delegates to the other modules.

[26] https://github.com/helmetjs/helmet

4.2.2 DB connector

This module is in the db-connector.js file and mediates the connection to the DB. Therefore, it uses node-postgres[27] with 10 number of clients in the pool (10 clients are permanently connected to the DB to avoid the delay while creating a new connection).

It just provides one function, `query`, which executes a query in the asynchronous way (do not block the execution of the entire application) and calls a callback provided by a parameter after the query is done.

4.2.3 Root file access

This module is stored in the file root-file-access.js and provides the functionality of getting all data about the frames. It uses `DB connector` to run the database queries for retrieving the data about the positions in the ROOT files and the precalculated statistics.

When it gets the data about the ROOT file where the desired data are stored, it passes them to the `Timepix frames accessor` module described in 4.2.3.1.

4.2.3.1 Timepix frames accessor

This module is implemented as the asynchronous Node.js addon and uses the `v8` interface built in Node.js. A have chosen the asynchronous way because the operations needed to get the desired data from the ROOT files are time consuming. If they were synchronous the response time to the requests could be really long with more users (is it visible even with 5 users using the integral mode).

The module also checks the incoming parameters for bad values. The parameters are:

- path to the ROOT file
- index of frame data
- index of clusters start
- index of clusters end
- callback function to call after the parsing is done
- final result callback
- final result structure
- appended data to pass some variables through

[27] https://github.com/brianc/node-postgres

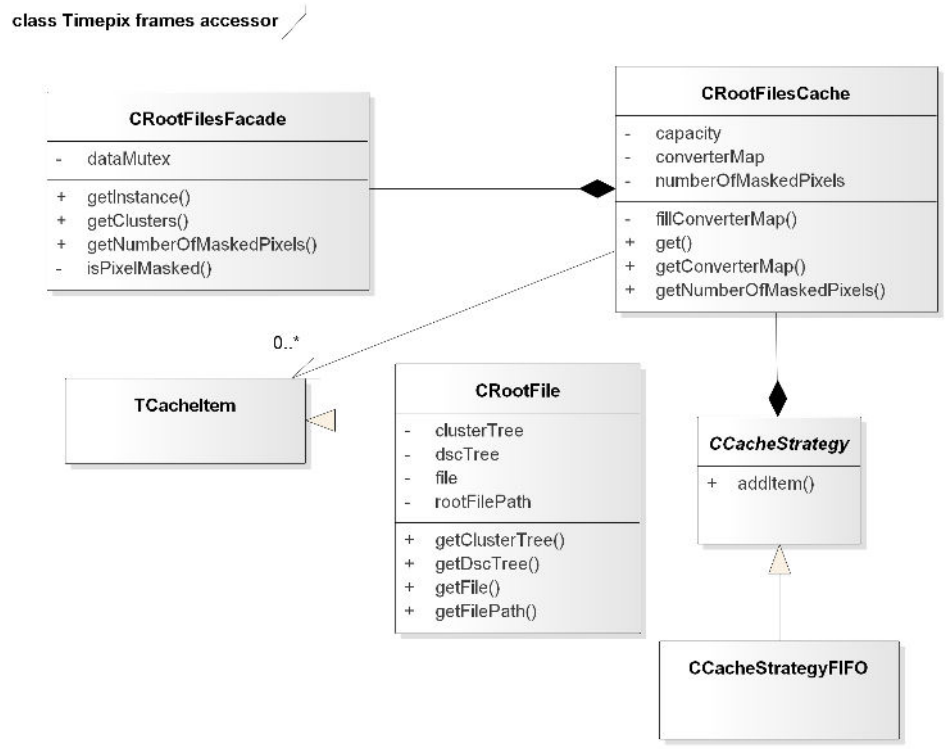

Figure 4.2: Class diagram of the Timepix frames accessor

I created a cache with opened ROOT files to speed the file accessing up. The class diagram of this addon is shown in the figure 4.2 and the classes are described below.

4.2.3.1.1 CRootFilesFacade This class provides the entry point to all the other classes. It is implemented as a singleton to provide only one existing instance. It creates `CRootFilesCache` and gives it `CCacheStrategy` to know how to add new items to the cache.

The function `getClusters` gets the desired data about the frame and its data about clusters. It parses the data and returns them as a string to be treated as JSON object.

4.2.3.1.2 CRootFilesCache `CRootFilesCache`, as the name suggests, is the cache for the ROOT files. It handles the files opening and closing, and the creation of converter maps (the maps with masked pixels).

4.2.3.1.3 CRootFile This class represents a cache item. It opens the given rootfile and keeps it open until it is destroyed by the cache.

4.2.3.1.4 CCacheStrategy It is an abstract class just to provide the needed interface for the other strategies. It defines just one method to be implemented - `addItem`.

4.2.3.1.5 CCacheStrategyFIFO It extends the `CCacheStrategy`, so it contains the implementation of the function `addItem`. FIFO is an acronym for "first in, first out" which in this context means that the fist element which got into the cache is the first to be removed when the capacity is reached.

4.2.4 Radiation timeline

This module provides all the information to generate the radiation timeline. It executes an SQL query and uses the `Timepix frames accessor` module to get the number of all masked pixels - this information is used to provide the right information about the matrix pixel occupancy.

4.2.5 Radiation map

Unlike the `Radiation timeline` module, this module does not need external information from the ROOT files. All desired data are stored or can be calculated from the DB. The SQL query is presented here to provide a closer look:

```
SELECT (energy/acq_time) AS energyflux
       (clusters_count/acq_time) AS particlesflux
       satellite_long AS long,
       satellite_lat AS lat
FROM frame
WHERE start_time > to_timestamp($1) AND
      start_time < to_timestamp($2)
```

The query gets all the frames and their statistics: energy flux (energy per second), particle flux (number of particles per second), and the frame location within the defined time range.

4.3 Client

By far the most lines of the code were created in this part of the application. Mainly the javascript part is described in this section. The HTML and CSS is trivial compared to JS code, so that I just mention them shortly.

The visualisation contains one HTML file. I use `div`s from the Bootstrap's grid to define the areas in the application. In all other parts I use the elements with the correct semantic value (`li` for menus and unordered lists in general, `dl` for definitions of variables, `table` for table data,...).

I divided CSS into 11 files representing areas in the visualisation (pixel matrix, radiation timeline,...), 1 file for variables and 1 file which imports all other for easy Sass compilation.

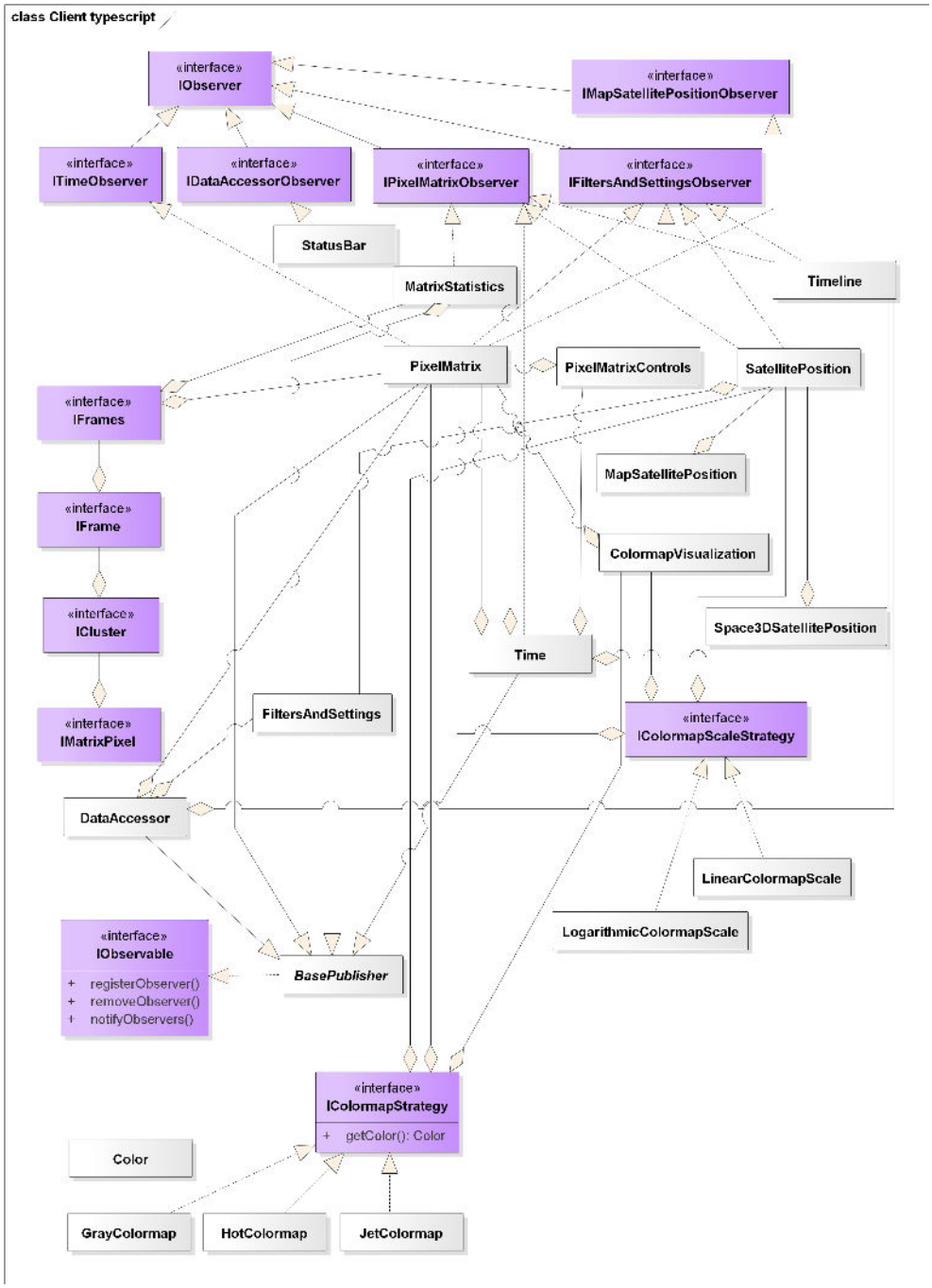

Figure 4.3: Class diagram of the client typescript code

The JS part is written in typescript and the class diagram is shown in the figure 4.3. This class diagram does not contain the attributes and functions to save space. It just shows the relationships of the most important classes and interfaces. I describe the crucial parts of the typescript code in the following sections.

4.3.1 Observer pattern

The observer pattern is used to notify dependent objects that the subject has changed his state [23]. It is widely used in the web applications for changes done by javascript.

I use observers for notifications of data arrival (`IDataAccessorObserver`), settings or filters (`ITimeObserver` for time, `IFiltersAndSettingsObserver` for changes in the main menu, and `IMapSatellitePositionObserver` for filtering by satellite positions). Each class which is a publisher (publishes its changes to the observers) extends the abstract class `BasePublisher`.

4.3.2 Data access

The data access is provided by the `DataAccessor` class. It is implemented as a singleton to ensure that just one instance is created. It is used by classes that needed the information from the server.

The `DataAccessor` uses the JQuery function `ajax()` to perform the needed requests of the client to the server API entry points defined in 4.2.1. It also measures the time of the requests to show it in the status bar.

4.3.3 Colours and colourmaps

Colourmap is a function that accepts value [0-1] and returns some colour representing the value. There are 2 points using the colourmap in the application: the pixel matrix 4.3.4 and the radiation timeline 4.3.6. While implementing the colourmaps I used a Matlab implementation and rewrote it to typescript.: hot (most used by space scientists because of its black background) - `HotColormap`, jet - `JetColormap`, and grey - `GrayColormap`.

Since the range of the counter values (energy) can span over 2 orders of magnitude (1- 2000), in a linear scale differences of low values can sometimes be hardly seen, so I implemented a logarithmic scale and possibility to switch between linear and logarithmic scale. This functionality is provided by `LinearColormapScale` and `LogarithmicColormapScale`.

4.3.4 Pixel matrix

The class `PixelMatrix` represents the pixel matrix of a frame. It uses 2 `canvas` elements for the visualisation to increase the speed of redrawing. The first one

is used to draw the background with clusters (tracks) and the second one is used to draw the overlay with the lines and the shadows while zooming.

The `canvas` element has attributes `width` and `height` set to 256 (the same as the pixel matrix dimensions) and the resizing is done via CSS width in %. The pixels are drawn individually using a 1x1 rectangle. There is a possibility to manipulate with one pixel in the `canvas` and a sharp mind could suggest that it should be faster than to use 1x1 rectangle. It is true, but only for some browsers. I tested the time needed to perform the actions for different browsers defined in in 2.3.2. The test results in the form of the numbers of operations per second are summarised in the table 4.1.

Method	Chrome 58.0.3029	Edge 14.14393.0	Firefox 52.0.0
1x1 rectangle	276,783	360,072	304,024
pixel manipulation	1,053,239	87,865	7,464

Table 4.1: Table of browsers test drawing 1x1 rectangle vs the pixel manipulation

For the Chrome and Edge the pixel manipulation is approximately 4 times faster. But the problem is with the Firefox browser. The pixel manipulation is 40 times slower than drawing the 1x1 rectangle and the number of operations in sec is 7,464 which is drastically low compared to the other numbers. This is the reason, why I used the drawing 1x1 rectangle instead of the the pixel manipulation.

The other important part of this class is the implementation of the data filtering and the statistics recalculating. I created a deep copy of the original data retrieved from the server (deep copy means that all information is copied as values not as references). Then I apply the created filters (mask/cluster attributes/...) on the copied data and calculate the statistics. When the user defines a new filter, I do not ask the server for the data again, but I use the saved original data instead.

The shown information of one pixel when mouse is across the pixel matrix are placed in a `div` element created by JS and moved with the mouse cursor. The text inside has 4 shadows (1 pixel to each direction) with the colour of the background defined by the colourmap to be nicely visible through the clusters.

Another interesting part is the implementation of the zooming of the pixel matrix. The zoom area is defined by 3 parameters: the zoom coefficient (to know how much bigger the pixels should be drawn), top left corner coordinates with bottom right corner coordinates to know what area should be drawn. While the user is dragging the mouse over the `canvas` elements the area which is then zoomed is surrounded by shadows (4 rectangles with some opacity) to highlight the zoomed area.

The pixel matrix statistics with histograms shown under the first menu item are handled by the class `MatrixStatistics`. All the statistics are calculated from the data retrieved from the pixel matrix (the already filtered data), so this class just loops over them and calculates all the statistics. The Google Charts are used to draw the histograms of the values - before redrawing a chart, the previous chart has to be manually removed by `clearChart` function to prevent a memory leak in the Google Charts library.

The controls to access the next and previous pixel matrices or to enter the animation mode are handled by the class `PixelMatrixControls`. The controls are disabled while loading a new frame. This class also handles the keyboard keys to manipulate with the GUI: left arrow to access the previous frame, right arrow to access the next frame, and the space bar to enter or leave the animation mode. The attribute `which` of the JQuery event passed to the handling function is used to identify which key was pressed.

4.3.5 Status bar

The status bar is handled by the class `StatusBar`. It shows if there are any queries to the server and their execution times. Thanks to be the observer of `DataAccessor`, the needed information are delivered to the class.

4.3.6 Radiation timeline

The radiation timeline is represented by the class `Timeline`. The Google Charts library is used to draw the chart and, as in the drawing of histograms, the previous chart has to be cleared before drawing a new one to prevent a memory leak.

The timeline supports energy, particles, and dose rates drawing around the displayed frame (the range is defined by the settings) and their fluxes (per second). The dose rates are energy multiplied by the constant defined in 3.5. While drawing the particle count or flux, the pixels occupancy is drawn too and its axix is shown on the right side of the graph. This functionality is achieved by a multiple series in the Google Charts graph.

4.3.7 Filters and settings

All the filters and settings that can be set by the user are handled by the class `FiltersAndSettings`. The JQuery references to the `input` elements holding the filters and settings values are stored within this class to speed up their retrieval. A list of interfaces is defined here to notify of the changes of the filters or settings - these interfaces are not shown in the class diagram to save space.

The class is also handling the filters minimum and maximum values defined in the configuration file and the values check to ease the user's work - so

he could not define a wrong value. This functionality is achieved by the JS functions `parseFloat` for the float values and `parseInt` for the integer values.

4.3.8 Satellite positions

The satellite positions are represented by the class `SatellitePosition`, which is internally using the classes `MapSatellitePosition` to show the position on 2D map and `Space3DSatellitePosition` to visualise it in 3D. This class handles the incoming data about the frames and sends the relevant information (positions and energy for the radiation map) to the other 2 classes to draw them.

4.3.8.1 2D map

The 2D map is represented by the class `MapSatellitePosition`. The Mercator projection defined in 3.4.1 is used to convert the longitude and latitude to 2D coordinates.

The map allows the filtering by the satellite positions which is achieved in the same way as the zooming in the pixel matrix, except that the map is not zooming and the defined shadows are drawn permanently until the positions restrictions are deleted or overwritten by other restrictions.

The information about the position is drawn inside a `div` element, created by JS and positioned to the top left corner. The interesting part is that the altitude of the satellite that is stored within the ROOT files and retrieved by the GUI is not the metres above the sea level, but the metres from the center of the Earth. So the real value has to be calculated by subtracting the Earth radius from the value got from the server.

4.3.8.2 3D visualisation

The 3D visualisation is handled by the class `Space3DSatellitePosition`. It creates the whole scene in the beginning.

The Earth mesh is created through Three.js by creating a `SphereGeometry` class. The material of the Earth is from 3 parts: texture map, bump map, and specular map. The texture map is the map of the world (this is used in the 2D map visualisation as well). The bump map is used to simulate bumps and wrinkles - on the Earth is means the mountains. The specular map is used to simulate reflection of the light from the seas and rivers. The clouds are simulated by creating a `SphereGeometry` slightly bigger than the Earth's geometry and putting the clouds image (with transparent areas) on it. Then the clouds rotation is achieved by rotating the whole mesh every time the scene is reloaded.

I used the square with different sides to show the satellite position, I was not satisfied with the visualisation, so I created my own model of the satellite from the available images of Proba-V. The satellite render image can be seen

in the figure 4.4. The position of the satellite is defined by the longitude, latitude, and altitude. The attitude of the satellite is is calculated by the matrix multiplication (Three supports it by calling the function `applyMatrix`). The matrices are defined directly from the definition in 3.4.2.

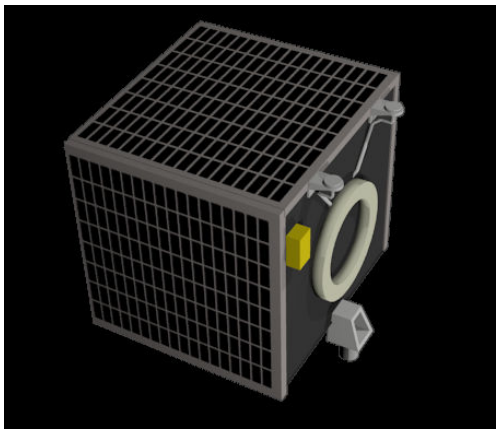

Figure 4.4: Satellite model in 3D

The Three extension `TrackballControls` is used to handle the scene rotation, translation and zooming.

Chapter 5

Testing

The application testing is presented in this chapter. There are many possibilities of the application testing: from the testing by the programmer or experts to the testing by end users. I have chosen the heuristic evaluation performed by experts and the usability testing performed by end users. The sources I have used are [24], [25], and [26].

5.1 Heuristic evaluation

One of the forms of testing without users is the heuristic evaluation. It is based on fulfilling heuristic's rules that differ with the used heuristics. It is performed by expert(s), whereby expert should not be the author of the application that is tested.

I used the most common heuristic - Nielsen's heuristic evaluation. It has 10 rules to fulfil (I just list them here, detailed information can be found in [24]):

1. Visibility of System Status
2. Match between a system and the real world
3. User control and freedom
4. Consistence and Standards
5. Error Prevention
6. Recognition Rather than Recall
7. Flexibility and Efficiency of Use
8. Minimalistic Design
9. Help users to recognise, diagnose, and recover from errors

10. Help and documentation

The important part is to choose the evaluators. I chose evaluators that were familiar with the web application development, so they had no problem with going through the rules.

Another decision of how many evaluators will be employed had to be made. The figures 5.1 and 5.2 helped me with it. I chose 3 evaluators - Bc. Lukáš Kořán, Bc. Petr Kubín, and Bc. Jan Růžička. All three of them were my fellow students, so they evaluated the application for free as a favour.

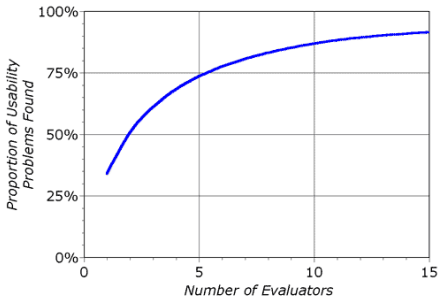

Figure 5.1: Application's problems found to the number of evaluators; source: [26]

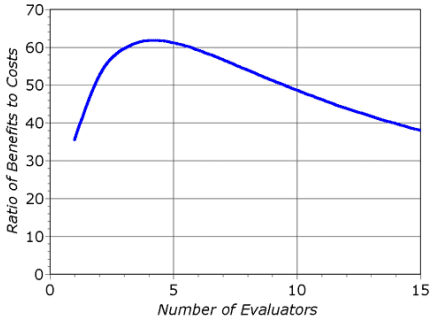

Figure 5.2: Ratio of benefits to costs to the number of evaluators; source: [26]

I list here the problems evaluators found and the solution I have made to remove the problem.

1. **Visibility of System Status**
   - **Status bar is too small to be well visible**:
     I made it bigger (from 70% to 90% of the normal font size)
   - **The loading gif stops when the radiation map is being calculated**:
     It happens because the javascript is calculating and drawing the map. I gave the text of what is happening to the loading gif, but I removed the text afterwards because it was not better. It is worth mentioning that it is happening in the older versions of browsers, so it could be removed automatically by the browsers in the future.
   - **The satellite disappears behind the Earth in the half of its orbit**:
     The user can fix the view to the satellite now.
   - **The state of the filters is not displayed. The user does not know if they are set or not**:
     I added a red border to the filters if they are set to visualise when the data are filtered.

- **Creating a mask from all shown pixels is slow (block application) on less equipped computers**:
  I removed the possibility to create mask from all shown pixels and added more suitable solution - the user can now enter a "mask creation mode" when the pixel matrix is not zooming, but the marked area now becomes the mask.

2. **Match between a system and the real world**
   No problems has been found - OK.

3. **User control and freedom**

   - **The undo possibility is missing when changing form inputs**:
     The modern web browsers support this functionality, so it will be possible for all users in the future.
   - **Filters can not be reset - only by the new load of the page or set them on default values**:
     I consulted this problem with physicists in IEAP and the filters resetting would be a nice addition, but is not necessary, so I did not implement it.

4. **Consistence and Standards**
   No problems has been found - OK.

5. **Error Prevention**

   - **There can be some values that are not possible to happen in the data in filters**:
     Some values can be set to negative even if it does not make sense (for example negative number of pixels in a cluster) if there was some problem during the calculation of their values, so the filters are left without any restrictions. I added some restriction to filters where this can not happen.
   - **One of the experts does not like the design and mentioned that he can be lost in the application in the way that he does not know which texts belongs to which part of the application**:
     This kinds of problem are really subjective, but there were some areas where, with a closer look, I could be lost too (without the knowledge of the whole application). So I doubled the width of the lines between main parts.

6. **Recognition Rather than Recall**

   - **There are no information that some filter is active**:
     I added a red border to the filters if they are set to visualise when the data are filtered.

7. **Flexibility and Efficiency of Use**
   - **The keystrokes are hidden from user (arrows for next/prev frame and space for animation mode)**:
     I added them to the information section.
   - **Filters and settings should be savable because of the frequent use**:
     I consulted this problem with physicists in IEAP and the filters resetting would be a nice addition, but is not necessary, so I did not implement it.
8. **Minimalistic Design**
   No problems has been found - OK.
9. **Help users recognise, diagnose, and recover from errors**
   - **The user that does not know about the problematic could have problem understanding all GUI possibilities**:
     I added some tooltips and help to the most of the functionality.
10. **Help and documentation**
    - **The application has an information section, but the "I" icon is too small**:
      I changed the icon to a bigger one.

To sum up the heuristic evaluation, the experts have identified possible issues, which were solved before the usability testing was performed to increase its quality.

5.2 Usability testing

Usability is the ease of use and learnability of a human-made object (in my case the final web application). It is not just about user satisfaction but about overall usefulness: efficiency, ease of use, and user friendliness. The usability testing is an attempt to quantify the software user-friendliness according to the following:

1. skill needed to learn the software
2. time required to become efficient in using the software
3. measured increase in user productivity (compared to a previous product if there is any)
4. a subjective assessment of a user's attitude toward using the software

The aim of the user testing is to improve or update an application, so it should answer how and what to update.

5.2.1 Users

I focused on the improving the application by observing the real users using it to find some usability problems. The usability testing was performed on the final application (after the heuristic analysis adjustments) in IEAP with the users who will be most likely using the application:

1. **Benedikt Bergmann** (age 30) - physicist, data analysis
2. **Stefan Gohl** (age 29) - researcher
3. **Thomas Billoud** (age 28) - PhD student, working on ATLAS-TPX (pixel detector)
4. **Petr Mánek** (age 22) - software engineer

All four of the testers were employees of IEAP and they were familiar with the pixel detectors. The major problems regarding the confusion of a new user, not familiar with the displayed, data were solved during the heuristic evaluation (it was done by application developers who knew nothing about the pixel detectors and their output data).

5.2.2 Scenarios

I created 8 scenarios (tasks) covering the functional requirements defined in 2.2 and some of the functionality I wanted to check for the usability. Every tester agreed to the process recording to provide better results of the test (all the videos are on the enclosed DVD). The scenarios and their process are:

1. **Display next/previous frame.**
   No problem - OK.
2. **Download the pixel matrix in the form of an image (png).**
   One tester had a problem with the transition to the download sub-menu to select the form of the download.
3. a) **Display frame from time 2014-11-20 12:25.**
      This task was a really big problem for 3 of the testers. They even did not find the functionality to perform it and had to ask me where it is in the GUI.
   b) **Display the histogram of the curly tracks volumes (keV).**
      No problem - OK.
   c) **Close the histogram.**
      No problem - OK.

   d) **Filter the curly tracks out and set minimum linearity to 0.5.**
   The task was done by all testers, but some of them filter all the tracks except the curly tracks. It was probably due to bad task formulation. Setting the linearity was not a problem. OK.

4. **Generate the radiation map of energy flux from 2014-12-01 to 2014-12-15 from frames with 0.2 acquisition times.**
   One of the testers set the wrong start date, but he found the right functionality - OK.

5. **Set another colormap for the pixel matrix.**
   No problem - OK.

6. **Change the radiation timeline range to 180 minutes.**
   One of the testers did not find the functionality, but he told me that he found it 5 minutes after the testing.

7. **Set the integral mode of the frames to 10 and display the arrow for the estimated particles direction (the arrow will show in the 3D visualisation). For better view zoom to the satellites in the 3D visualisation.**
   No problem - OK.

8. **Mask out (create a mask) some rectangle area in the top right corner of the matrix.**
   It took a while to get the functionality, but all users successfully completed this task in the end - OK.

5.2.2.1 Evaluation

The biggest problem was to find the functionality to display frame from a certain time. There is the time of the frame above the pixel matrix shown in the GUI, but it does not provide the requested functionality. It is in the filtering section in the controls. The possibility to solve this problem was to add the functionality to the displayed time of the frame (all of the the users were clicking on it) or just to open the time picker when the user click on the time. I chose the second option, so now when the user clicks on the time of the frame, the time picker shows up.

The problem of finding the form for the settings for the time range (task 6) were caused by the users not familiarising themselves with the GUI. The users were told to try out the application before going to the tasks. For my surprise none of them took this possibility and went straight to the tasks without any knowledge of the GUI. The testers told me that they found the functionality immediately after the testing, so I did not perform any steps to correct this problem.

Some minor issues could be identified by studying the video recordings. These findings triggered the following improvements:

- a possibility to select/deselect all cluster types in the filter
- bigger steps (5%) in the radiation map generation form
- more explanation text to the "number of pixels" in the radiation map generation form
- the scrolling affected the datetime pickers (they were scrolling too), so I added a feature that hides the pickers when the user scrolls
- Czech keyboard has "," not "." in the numeric keyboard section and the application was treating numbers like 5,5 as just 5, so I improved the functionality to work on these numbers too

5.2.3 Questionnaire

A questionnaire was given to the testers after completing the tasks. It contained 5 simple questions mainly to find out what to improve or what is good enough in the current state:

1. **Were you able to perform all tasks? If not, what task was not performed?**
   One tester did not complete task 6 (but he found the functionality 5 minutes after test). All other testers completed all tasks successfully.
2. **What did you like about the visualisation?**
   The response time and 3D visualisation was appreciated.
3. **What did you not like about the visualisation?**
   The testers wanted to have more information about the functionality build in the GUI. Some controls were hard to find for them.
4. **What existing functionality would you upgrade or do the different way (for example add histograms of another type)?**
   I had got a lot of response here:
   - Allow histogramming of all variables saved in the ROOT file as precalculated values, i.e. linearity, clstrHeight_keV, ... And add these values to the filter as well.
   - I would like to have more options for the timeline (e.g. same filters you have for the histograms, the angles, number of tracks, number of the different types of the tracks)
   - A possibility to filter on the angles as well. So you may know which particular direction occurs where on earth or when.

- More navigation possibilities for the 3D map, to move earth away from the center of the image.

5. **What functionality would you add to the visualisation?**
   I had got a lot of response here too. Some of it was overlapping the previous question:

   - I would put some more information to it. Something like a little clickable i that drops down some small text window with an explanation of the functionality.
   - Maybe add a click (e.g. center click of mouse) for translation on the 3D visualisation.
   - Option to manipulate panel width, perhaps make them temporarily full screen.

5.2.3.1 Evaluation

A good thing is that testers appreciated the response time of loading a new frame. They wanted more information about the functionality in the GUI. Therefore I added some labels to the form inputs. The added/upgraded functionality comes directly from the answers. I added more types of histograms and more variables in the filter and the translation of the scene was added to the 3D visualisation. One of the testers wanted the possibility to show the pixel matrix in full screen - I already implemented this functionality, but then I removed it because the pixel matrix was approximately the same size in the full screen mode as in the normal mode.

Conclusion

The goal of the thesis was to create a web application to visualise the data acquired by the SATRAM/Timepix pixel detector onboard the ESA Proba-V satellite operating in Low Earth Orbit. The analysis, design, and implementation of the application was done and the heuristic evaluation with user testing followed the implementation.

The thesis uses various algorithms and methods to perform the desired functionality. The Mercator projection was used to show the positions of the spacecraft on 2D map and the satellite attitude was determined with the use of the quaternions.

To create the whole application, many techniques such as C++ programming, web backend and frontend development, or graphical user interface design had to be used. The result is a complex and modular application where each of its parts can be easily improved, changed or reused in other application, if need be.

The application is now used by the physicists in the Institute of Experimental and Applied Physics, Czech Technical University in Prague to provide a closer look at the acquired data. There are many possibilities for the improvements such as the radiation map in the 3D visualisation or the radiation timeline enhancement in the future.

Bibliography

[1] Granja, C.; Polansky, S.; et al. The SATRAM Timepix spacecraft payload in open space on board the Proba-V satellite for wide range radiation monitoring in LEO orbit. *Planetary and Space Science 2016*, 2016, [cit. 2016-11-22]. Available from: `http://dx.doi.org/10.1016/j.pss.2016.03.009`

[2] Granja, C.; Polansky, S.; et al. Mapping the space radiation environment in LEO orbit by the SATRAM Timepix payload on board the Proba-V sattelite. *Conference: THERMOPHYSICS 2016: 21st International Meeting*, 2016.

[3] Granja, C.; Polansky, S.; et al. Quantum Imaging Monitoring and Directional Visualization of Space Radiation with Timepix based SATRAM Spacecraft Payload in Open Space on board the ESA Proba-V Satellite. *Conference: 2014 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, 2014.

[4] Granja, C.; Polansky, S.; et al. Directional Visualization of Space Radiation Quanta with Timepix based SATRAM Payload on-board ESA Proba-V Satellite. *Proceedings of Science PoS X LASNPA 003*, 2014.

[5] Jakůbek, J. Pixel device TimePix. [cit. 2016-11-24]. Available from: `http://aladdin.utef.cvut.cz/ofat/others/Timepix/`

[6] Štěpán Polanský; et al. Medipix and Measurements at Seibersdorf in the Framework of ARDENT, lecture.

[7] Bergmann, B.; Caicedo, I.; et al. ATLAS-TPX: a two-layer pixel detector setup for neutron detection and radiation field characterization. *IOP Publishing for Sissa Medialab*, 2016.

[8] CERN. *ROOT documentation [online].* [cit. 2017-01-04]. Available from: https://root.cern.ch/root/htmldoc/guides/primer/ROOTPrimer.html

[9] Mlejnek, J. Analýza a sběr požadavků, summer semester 2014, lecture [Online] [cit. 2016-12-15]. Available from: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/03/03.prednaska.pdf

[10] Mlejnek, J. Analýza problémové domény, summer semester 2014, lecture [Online] [cit. 2016-12-15]. Available from: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/04/04.prednaska.pdf

[11] W3Schools. Media Queries. 2016, [Online] [cit. 2016-12-15]. Available from: http://www.w3schools.com/cssref/css3_pr_mediaquery.asp

[12] Valenta, M. Evaluation of SQL, summer semester 2016, lecture [Online] [cit. 2016-04-12]. Available from: https://edux.fit.cvut.cz/courses/MI-PDB/en/lectures/03/start

[13] Garrett, J. J. The elements of user experience: User-Centered Design for the Web and Beyond, Second Edition. 2011.

[14] Žikovský, P. Návrh UI, prototypy, winter semester 2016, lecture [Online] [cit. 2016-04-13]. Available from: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x02-navh_a_prototyping.pdf

[15] W3Schools. Choosing the right doctype for your HTML documents. 2016, [Online] [cit. 2017-04-13]. Available from: https://www.w3.org/wiki/Choosing_the_right_doctype_for_your_HTML_documents

[16] Frees, S. Getting your C++ to the Web with Node.js. 2015, [Online] [cit. 2017-04-18]. Available from: https://nodeaddons.com/getting-your-c-to-the-web-with-node-js/

[17] Osborne, P. The Mercator Projections. 2013, [cit. 2017-04-18]. Available from: https://zenodo.org/record/35392/files/mercator-15dec2015.pdf

[18] Gohl, S. Satellite Orientation Determination for Proba-V, 2016, unpublished article.

[19] ESA. Technical report: [EN-22] PROBAV-TN-00159-VE F1 Reference Frames and Control Errors, technical report.

[20] ICRP. The 2007 Recommendations of the International Commission on Radiological Protection. *Annals of the IRCP*, 2007.

[21] Špaček, P. Principles of OO Design, winter semester 2016, lecture [Online] [cit. 2016-04-26]. Available from: https://edux.fit.cvut.cz/courses/MI-DPO/_media/lectures/mi-dpo-01-principles-of-ood.pdf

[22] Špaček, P. Facade – Mediator – Singleton, winter semester 2016, lecture [Online] [cit. 2016-04-27]. Available from: https://edux.fit.cvut.cz/courses/MI-DPO/_media/lectures/mi-dpo-03-facade-mediator-singleton.pdf

[23] Špaček, P. Iterator – Composite – Builder – Observer, winter semester 2016, lecture [Online] [cit. 2016-04-27]. Available from: https://edux.fit.cvut.cz/courses/MI-DPO/_media/lectures/mi-dpo-04-iterator-composite-builder-observer.pdf

[24] Žikovský, P. Testing without users, winter semester 2016, lecture [Online] [cit. 2016-04-21]. Available from: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x03-semestralka_testing_without_users.pdf

[25] Žikovský, P. Usability, Special Testing, Personas, winter semester 2016, lecture [Online] [cit. 2016-04-21]. Available from: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x04-usability_special_testing_personas.pdf

[26] Nielsen, J. How to Conduct a Heuristic Evaluation. 1995, [cit. 2017-04-21]. Available from: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

Appendix A

Acronyms

API Application Programming Interface

Ajax Asynchronous JavaScript and XML

CERN European Organization for Nuclear Research

CSS Cascading Style Sheets

CTU Czech Technical University

DB Database

ERA Earth Rotation Angle

GPU Graphics Processing Unit

GUI Graphical user interface

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IEAP Institute of Experimental and Applied Physics

JS Javascript

LEO Low-Earth Orbit

SATRAM The Space Application of Timepix based Radiation Monitor

UML Unified Modelling Language

Appendix B

Contents of enclosed DVD

- `readme.txt` ..... the file with DVD contents description
- `analysis-and-design` ..... the analysis and design
- `samples` ..... screenshots of the final application
- `testing` ..... videos and data from the user testing
  - `heuristic-evaluation` ..... the heuristic evaluation
  - `user-testing` ..... the user testing videos and scenarios
- `src` ..... the directory of source codes
  - `server` ..... implementation of the server sources
  - `data-preprocessor` . implementation of the data preprocessor sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `MT-Herbert-Waage-2017.pdf` ..... the thesis text in PDF format