UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Rodion Krjutškov

Camera Component for the ESTCube-2 Mission Control System

Bachelor's Thesis (9 ECTS)

Supervisor: Dr. Andris Slavinskis
Co-supervisor: Alo Peets

Tartu 2018

Camera Component for the ESTCube-2 Mission Control System

Abstract:

ESTCube-2 is the second satellite project of the Estonian Student Satellite Programme. Its main scientific mission is to test the plasma break technology on the low Earth orbit. The hardware and the software for the project are developed by students. Computer software called Mission Control System is being developed to monitor and control the satellite after launch. The system consists of multiple components and applications connected in a microservice architecture. This paper focuses on the research and development process of the Mission Control System camera component. The author describes challenges related to displaying the satellite images in a modern web application, analyses possible solutions and provides an overview of the software prototype implementation.

Keywords:

ESTCube-2, space technology, Flexible Image Transport System (FITS), web software development, microservice architecture, front-end application

CERCS: P175, Informatics, systems theory

Kaamera komponent ESTCube-2 missiooni juhtimissüsteemi jaoks

Lühikokkuvõte:

ESTCube-2 on teine Eesti Tudengisatelliidi Programmi satelliidi projekt. Selle peamiseks uurimisülesandeks on plasma piduri tehnoloogia testimine maa orbiidil. Projekti riistvara ja tarkvara arendavad üliõpilased. Arendamisel on tarkvara nimega missiooni juhtimissüsteem, mis aitaks satelliiti pärast orbiidile saatmist jälgida ja juhtida. Süsteem koosneb mitmest osast ja rakendustest, mis on mikro-teenuste arhitektuuri sees ühendatud. Käesoleva lõputöö autor kirjeldab väljakutseid, mis on seotud satelliidi piltide kuvamisega kaasaegses

veebirakenduses, analüüsib võimalikke lahendusi ja annab ülevaate tarkvara prototüübi arenduse protsessist.

Võtmesõnad:

ESTCube-2, kosmosetehnoloogia, Flexible Image Transport System (FITS), veebi tarkvara arendus, mikroteenuste arhitektuur, front-end rakendus

CERCS: P175, Informaatika, süsteemiteooria

Table of Contents

1 Introduction

In the fast-changing field of Software Engineering, new technologies and trends appear on the regular basis. Companies invest significant funds in engineering to deliver new releases frequently and provide necessary software updates. One of the major development trends of the past decade is the increasing popularity of web-based and cloud-based solutions. Such approach allows software to be platform-independent, removes hardware restrictions for the user's machine and allows instantaneous and seamless software updates.

When the Estonian Student Satellite Foundation started preparations for their second project, they faced numerous software-related challenges. The software used in their previous mission, ESTCube-1, was not compatible with the ambitions of their second mission, ESTCube-2. The team set out to develop new hardware and software, including Mission Control System, a unified web-based solution for satellite interaction. The system consists of multiple microservices, or components grouped by functionality.

The present work introduces the camera component of the Mission Control System and describes the research and development process. The component is going to provide functionality required to browse the satellite images and issue camera-related commands. The images on the satellite are saved using the Flexible Image Transport System (FITS) standard, which is not used commonly in web-based applications. To handle these files in the Mission Control System, it is necessary to analyse existing open-source software libraries that were created by the community.

The paper consists of three major parts:

- the Background chapter introduces the missions of the Estonian Student Satellite Programme, describes the existing solutions and states the problem;
- the Camera Component Analysis chapter describes the process of initial research, specifies the functional requirements and presents the proposed solutions;
- the Camera Component Prototyping chapter follows the development process, describes the methods, gives examples of concrete problems and solutions that were implemented.

2 Terms and Notations

- **Mission Control System (MCS)** is a software system that is being developed for theESTCube-2 project. The purpose of the system is to monitor and control the satellite after launch.
- **Web client** is an application that communicates with a web server. Usually refers to the web browser on a user's computer.
- **Web server** is a computer system that processes and fulfils requests of web clients.
- **Application Programming Interface (API)** is a set of clearly defined methods of communication between a web client and a web server.
- **Graphical User Interface (GUI)** is a mean of human-computer interaction that allows a person to interact with computer software through visual indicators and icons.
- **Thumbnail** is a small-resolution digital image that is used as a preview of a larger digital image.
- **Pixel** is a smallest controllable element of a digital image presented on a screen.
- **Website wireframe** is a visual guide that is used to present website structure and layout.
- **Hypertext Transfer Protocol (HTTP)** is a request-response application protocol of the client-server software system. It is the foundation of data communication over the internet.
- **Hypertext Markup Language (HTML)** is the standard language used to create web pages. Describes the structure of the page semantically.
- **Joint Photographic Experts Group (JPEG)** is a commonly used graphics file format that allows lossy data compression. The degree of compression can be adjusted, resulting in adjustable file size.
- **Portable Network Graphics (PNG)** is a graphics file format that allows lossless data compression. Most commonly used lossless image compression format used on the internet.
- **Flexible Image Transport System (FITS)** is a digital file format that is used for transmission, processing and storage of scientific and other images. Most commonly used digital file format in astronomy.

- **Header-Data Unit (HDU)** is a data structure inside a FITS file that consists of a header and the data the header describes.
- **Node Package Manager (NPM)** is a software that consists of a command line client and an online database of software packages, called the registry. Used to install and manage JavaScript software packages.
- **Raw image data** is a minimally processed data from the image sensor of a digital camera.
- **Red, blue, green, alpha (RGBA)** is a combination of the red-blue-green color model with an alpha channel information that describes the image transparency.
- **Document Object Model (DOM)** is an application programming interface that describes HTML documents as a tree structure.
- **Goddard Space Flight Center (GSFC)** is a NASA space research laboratory for developing and managing unmanned scientific spacecraft.
- **High Energy Astrophysics Science Archive Research Center (HEASARC)** is a NASA center for managing and archiving of high energy astronomy data.

3 Background

3.1 Small Satellites

The miniaturized satellite industry is developing rapidly in recent years. According to the classification, small satellites are satellites with weight under 500 kilograms [1, p. 5]. The main reason for making satellites smaller is significant cost reduction. Such satellites can be launched using smaller and cheaper launch vehicles or even delivered to orbit in excess capacity of larger launch vehicles.

Nanosatellite is a small satellite with weight from 1 to 10 kilograms [1, p. 5]. Example applications of nanosatellites include scientific and technology research, education, earth observation, astronomy and military purposes. In order to further reduce the costs and development time, the CubeSat Project was started, and a standard for design of nanosatellites was created. CubeSat is a 10-centimeter cubical satellite unit with a mass up to 1.33 kilograms [2, p. 5].

3.2 First Estonian Satellite

As described in a publication by Slavinskis et al. [3], ESTCube-1 was a project led by University of Tartu and supported by European Space Agency. In course of the project a group of students developed a CubeSat satellite in collaboration with international partners. The scientific mission of the satellite was to conduct the first in-orbit experiment with electric solar wind sail technology.

ESTCube-1 was launched on May 7, 2013. The mission was mostly successful, and the satellite operated as expected, except several issues. All the subsystems of the satellite were developed in-house by a team of students with no previous experience. The team gained valuable insights, and Estonian Student Satellite Foundation was established with a goal to plan and develop next missions.

3.3 ESTCube-2 Mission and Goals

ESTCube-2 is an ongoing project developed primarily by students of University of Tartu. The aim of the project is to create a new nanosatellite and launch the next mission that is designed based on the outcomes of the previous mission.

As described in a paper by Ehrpais et al. [4], the main scientific mission behind ESTCube-2 satellite is to conduct further testing of electric solar wind sail and plasma break (also known as Coulomb drag propulsion). This technology was previously subject of experiments conducted in the missions of ESTCube-1 and Aalto-1 satellites.

3.4 Mission Control System

In order to communicate with a satellite and control the command flow between the satellite and a ground station, special computer software is required. Several applications and application expansion modules were developed by members of the Estonian Student Satellite Programme and used successfully in the ESTCube-1 project. This software was developed in different programming languages (Python, C), to be executed directly on the satellite operator's workstation.

In the ESTCube-2 project a set of such software constitutes the Mission Control System (MCS). In effort to create a modern, platform-independent, scalable software solution, ESTCube-2 team decided to develop the applications in a web-based microservice architecture. Such approach allows components to be executed in an isolated environment, transmitting data between each other over the network using Application Programming Interfaces (APIs). The components could be defined based on the principal functionality, programming language used in the application or other criteria.

It is a common practice in web software development to clearly separate server-side applications from client-side applications [5, p. 149]. The former ones are also referred to as back-end applications, and the latter ones as front-end applications. Back-end applications are executed on the server and therefore have direct access to predictable amount of hardware resources and to the file system. Front-end applications run on the user's computer, inside the environment of a web browser. It is, therefore, impossible to predict the amount of available hardware resources and access the local file system.

ESTCube-2 Mission Control System consists of one front-end application and multiple back-end microservices. The purpose of the front-end application is to provide the satellite operator with means to view the satellite data, interact with this

data and issue control commands to the satellite. This functionality is supported by the back-end applications that accept user's input transmitted from the front-end, execute server-side commands, have means to access the data and pass commands to the satellite.

3.5 Previously Used Camera Software

Among other hardware and software developed during the ESTCube-1 project, an application called PyCAM was created to handle camera-related tasks. The application was implemented in Python programming language and provided an integrated graphical user interface. It was an all-in-one solution that allowed satellite operator to browse and view the satellite images, inspect the image metadata, set image download options, run integrated tests of the satellite camera. Some software libraries used in this application are currently deprecated.

3.6 Problem Statement

Taking into consideration the intention to create a completely new version of the Mission Control System, based on the web-based approach and the software architecture inferred from such approach, it is necessary to conclude that it is not possible to fully reuse the programs that were developed for the ESTCube-1 mission. It is therefore necessary to analyse strategies for replacement of this software with updated subsystems, define and implement components as microservices.

4 Camera Component Analysis

4.1 Definition

The name camera component describes a subset of Mission Control System that provides satellite operator with functionality required to view images created by the satellite camera, inspect the image metadata and use the graphical user interface to issue commands to the satellite.

4.2 Requirements Engineering

In software engineering, the cornerstone of the development activity is the part called requirements engineering. It is a process that consists of requirements gathering, requirements analysis and requirements specification [6, p. 68]. The end goal of a requirements engineering effort is to gain better understanding of the problem and formalise a list of functional requirements for the future software. These requirements can change in the course of the development process, but the initial record still serves as a strong starting and reference point.

In order to gather functional requirements for the camera component, several interviews with ESTCube-2 team members were conducted. The people directly involved in the past and the present development of the satellite camera, as well as persons likely to be involved in the satellite operator role for ESTCube-2 mission, answered a series of questions about their understanding of the system and their expectations regarding interaction with the Mission Control System camera component in the future. Following these interviews and discussions, it became possible to compile the initial functional requirements list.

Table 1. Initial functional requirements for the camera component

#	**Requirement**	**Priority**
1	Display an image received from the satellite	Critical
2	Trigger acquisition of an image by the satellite	High
3	Zoom in/out the selected image	High
4	Pan the selected image	High
5	Rotate and flip the selected image	High
6	Select a region of interest on the image	High
7	Display a list of images stored on the satellite and in the Mission Control System	High
8	Save the selected image using PNG or FITS standard	High
9	Switch between colour channels of the selected image	Medium
10	See thumbnails of satellite images	Medium
11	See a histogram of the selected image	Medium

Presented in Table 1, the requirement #1, "display an image received from the satellite" was assigned a critical priority level because it is not possible to work on development of any other functionality of the camera component until this requirement is fulfilled. Meeting this requirement also provides the biggest technical challenge, as described in the section Displaying the Images.

Some functional requirements were revisited and modified in the development process, as a progress was made in various parts of the system and new decisions and realisations happened between parties involved.

Table 2. Additional functional requirements for the camera component

#	Requirement	Priority
12	Plot the full-size images on top of thumbnail images	Low
13	Display original pixel value on hover/click	Low
14	Display average/high/low pixel value for region of interest	Low

Having formalised the functional requirements, it is possible to proceed with the architectural definitions and first iterations of prototyping.

4.3 Architecture

Mission Control System development team decided to provide the graphical user interface in the form of a dashboard which consists of multiple subsystem tabs. Each tab presents satellite operator with a clearly defined subset of the MCS functionality. It was therefore suggested to develop the camera component front-end part in a such way that would make it possible to integrate it into one dashboard tab in the future. The website wireframes were created to illustrate how the camera component features are expected to fit into the dashboard view.

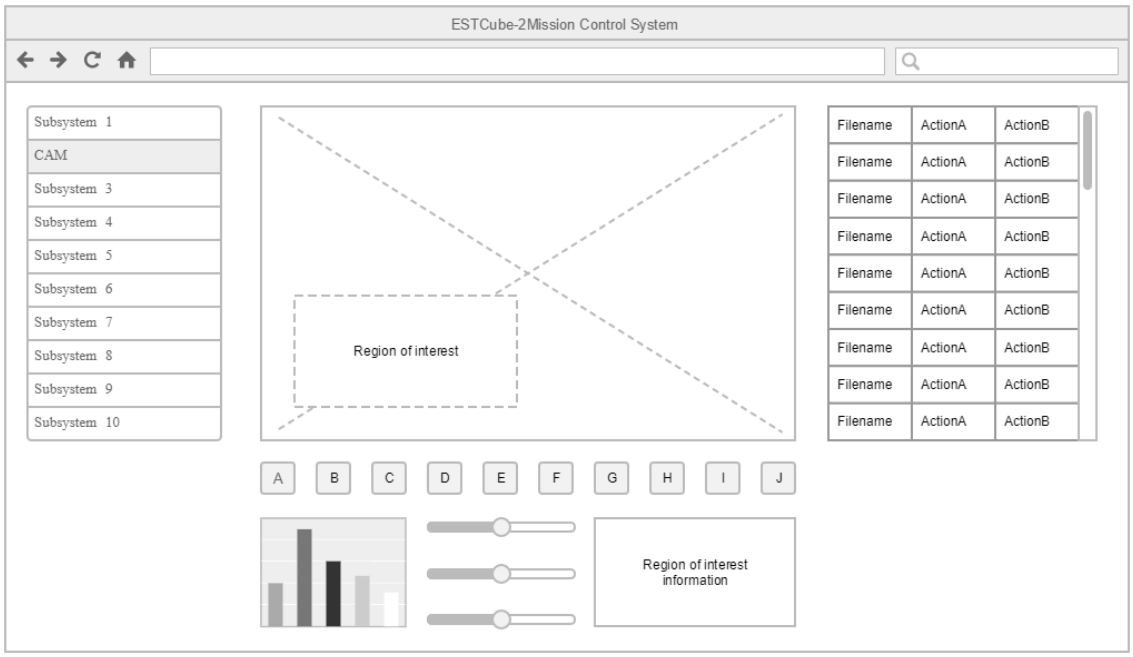

Figure 1. Website wireframe of the camera component in the MCS dashboard

The back-end applications developed in a micro-service architecture do not necessary need to be merged into one project, as they can communicate between each other over the APIs. Using the same approach in the front-end application would create unnecessary replication of dashboard elements and logic over all the subsystems. Therefore, to allow effortless integration in the future, extra effort must be dedicated to developing the parts of the front-end application, since the code base needs to be merged with the work of other developers into a single project.

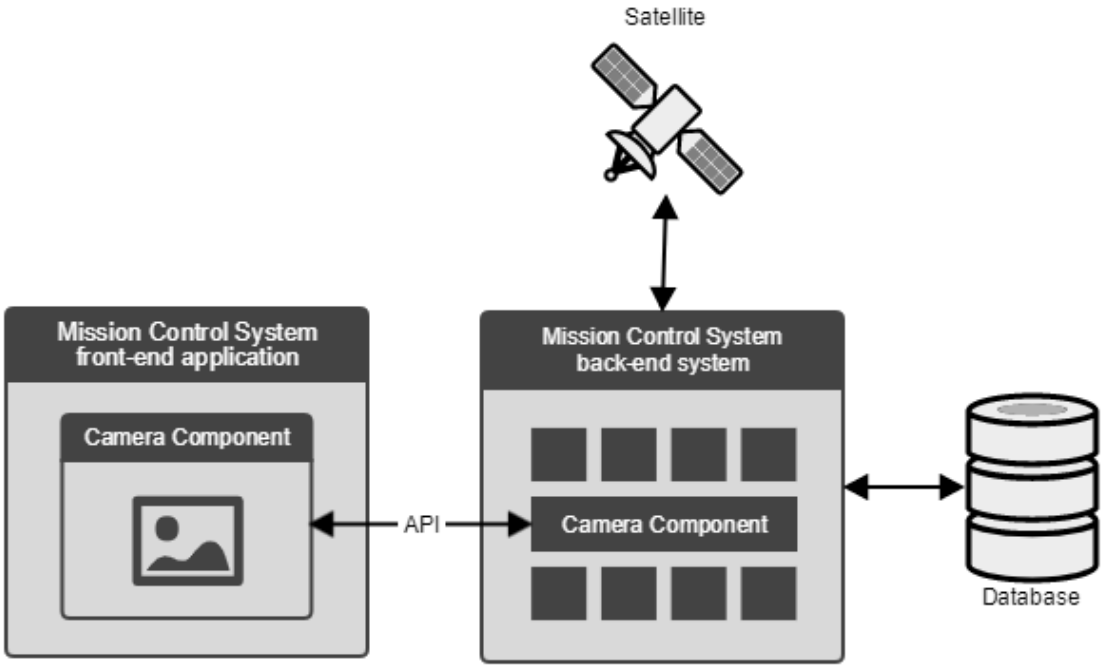

Figure 2. Camera component architecture in the MCS context

4.4 Displaying the Images

To display the images sent from the satellite, it is necessary for the camera component software to read the image data from the files saved using certain file standards. One of these standards, the Joint Photographic Experts Group (JPEG), is used widely for encoding of digital images. Such images are saved using a “lossy” compression algorithms, therefore do not preserve the original pixel values. In the ESTCube-2 software the JPEG compression is used to create the camera image thumbnails that provide a preview of the original full-size images, allowing more efficient data transfers. Displaying a JPEG image in a web interface is a routine task that typically does not require knowledge of the standard definitions.

The second file standard that is used in the ESTCube software, on the other hand, does not appear frequently on the web. The Flexible Image Transport System (FITS) is a data standard that is designed specifically for use in astronomy. The standard allows transport, analysis and archival storage of scientific data sets [7]. The first version of FITS was developed in 1981, and the latest release of the standard is version 4.0 that is described in the FITS Standard Document, published on 22 July 2016 [8].

A FITS file consists of several FITS structures: a primary header and data unit (HDU), optional extensions, optional special records. The header contains the image metadata in a human-readable key-value format. A Basic FITS file consists of a single (also known as primary) header-data unit that may contain a single image data array. Each value in such array represents an image pixel value. It is a common practice to apply a scaling function to the data in the array because of the different methods used for integer value representation. To extract the real values, the `BZERO` and `BSCALE` keyword values must be extracted from the header and applied to the data using the following formula:

```
physical value = BZERO + BSCALE × array value
```

Unlike JPEG, FITS files can't be displayed simply in a web browser, as such support is not integrated into the browser software. The FITS Support Office at NASA/GSFC compiled a list of software libraries and packages written by enthusiasts in many programming languages. The functionality varies between libraries, some of them provide only basic methods for reading the image data, some allow writing and working with multidimensional data tables.

Taking into consideration the separation between the back-end and the front-end applications imposed by the Mission Control System architecture design, it is necessary to address an important development problem. Since the images in FITS standard cannot be displayed to the user in the browser, the FITS image data must be converted into another format suitable for display.

4.5 Image Data Operations

The conversion form one data format to another can be performed both in the front-end and in the back-end applications. As the images stored in ESTCube-2 FITS files have a maximum resolution of 2752 by 2004 pixels, or 5 515 008 pixels, a one-time operation to read and manipulate these values could require a significant amount of hardware resources.

Delegating such task to the back-end application would minimise the load on the client's machine, but it could potentially create a high load on the mission servers in a situation when many clients would connect simultaneously, sending multiple requests to the same API endpoint. Such scenario cannot be ruled out, especially because some parts of the Mission Control System API may be made accessible to the public in the future. Eventually, the excessive load on the server would have to be addressed by a software scaling, load balancing and network security solutions.

Delegating the data conversion task to the front-end application would impose certain requirements to the user's computer hardware, specifically on the amount of available Random-Access Memory (RAM) and the Central Processing Unit (CPU) performance.

Both approaches were studied and prototyped in the following phase of development.

5 Camera Component Prototyping

5.1 Methods

Agile software development refers to a widely accepted approach in software engineering. According to a survey, the success rate of agile methods is much more positive than that of classical project management [9]. Among the core principles of agile software development are:

- early and continuous delivery of valuable software;
- delivering working software frequently;
- welcoming changing requirements [10].

Following these principles, the Mission Control System development team creates software prototypes, some of which are later discarded in favour of a better solution. The same strategy was used in the development of the camera component.

5.2 Initial Development

The first Mission Control System dashboard prototype was implemented using Angular, a front-end web application platform maintained by Google [11]. No significant camera component functionality was developed in this iteration, and in few months the entire prototype was replaced with a new project based on React, a JavaScript library for building user interfaces, which is developed and maintained by Facebook [12]. The reasons for switching from Angular to React included:

- better tooling and development environment;
- smaller code base, therefore smaller data transfers;
- bigger and more active development community [13].

Another clear advantage of the React library for the Mission Control System GUI development is its component-based nature. React components are encapsulated and therefore can be easily moved between projects, supplemented only with a formal list of external dependencies. Following this approach, the camera component application was set up in a separate project, with an aim to develop the React components for future integration. The project has minimal dependency on the current state of development in other parts of the MCS.

5.3 The React Project Setup

JavaScript is the only programming language that is supported by all modern web browsers. It is executed by a JavaScript engine that is built into the web browser, therefore it does not require installation of any plug-ins. JavaScript can be used to make web pages interactive by manipulating the pages content [14].

A conventional modern front-end application makes use of external JavaScript packages [15]. Packages can be installed into the project using software such as Node Package Manager (NPM). A special file called *package.json* is created inside every project to store, among other important metadata about the project, the list of external dependencies. As every direct package dependency can infer multiple indirect package dependencies that are required for correct work, a dependency tree is created and saved by the package manager in an auto-generated lock-file. Dependencies in the camera component front-end application are managed with help of Yarn, a package manager software developed by Facebook [16].

The React library allows developer to organise code in components that are written using special syntax called JSX. Every component declares explicitly how it is meant to be rendered in the web browser. Components can transfer data between each other using component properties and state. To create a production software distribution, it is necessary to compile the JSX syntax into conventional JavaScript. To simplify the task of setting up and managing the development process of a project based on the React library, we used a tool called create-react-app [17]. The application can be served for local development with an integrated HTTP server using only two commands:

```
yarn
yarn start
```

A production distribution can be created by running the following commands:

```
yarn
yarn build
```

After running the above script, the *build* folder in the project will contain a compiled and minified version of the code that can be served to the user using an HTTP server.

5.4 Development of the React Components

Based on the functional requirements described in the section 4.2 and on the initial wireframes such as the ones presented in the section 4.3, we focused our effort on development of two React components:

- a canvas component that can display satellite images (requirement #1) and provides the functionality for the image manipulation (requirements #3, #4, #5, #6);
- a gallery component that can display a list of images stored both on the satellite and in the Mission Control System (requirement #7) and provides means to trigger image-related commands (requirements #2, #8).

These two components are grouped under a parent component that describes the positioning of elements relative to each other. The parent component also plays an important role of the data flow controller within the application. The following scenario describes the internal workflow triggered when the application is first loaded into the user's browser:

1. The parent component sends an HTTP GET request to the backend application in order to retrieve the list of the files from the Mission Control System.
2. The parent component passes the received data through the properties to the gallery component.
3. The gallery component renders a list of the images and maps the function buttons in the GUI to correct back-end application API requests.

When the user selects an image to display from the galley list, another scenario unfolds:

1. The click event is passed up from the gallery component to the parent component through the properties.
2. The parent component sends an HTTP GET request to the backend application in order to retrieve the selected image data.
3. The received data is processed in the parent component and then passed to the canvas component through the properties.
4. The canvas component renders the image in the image area.

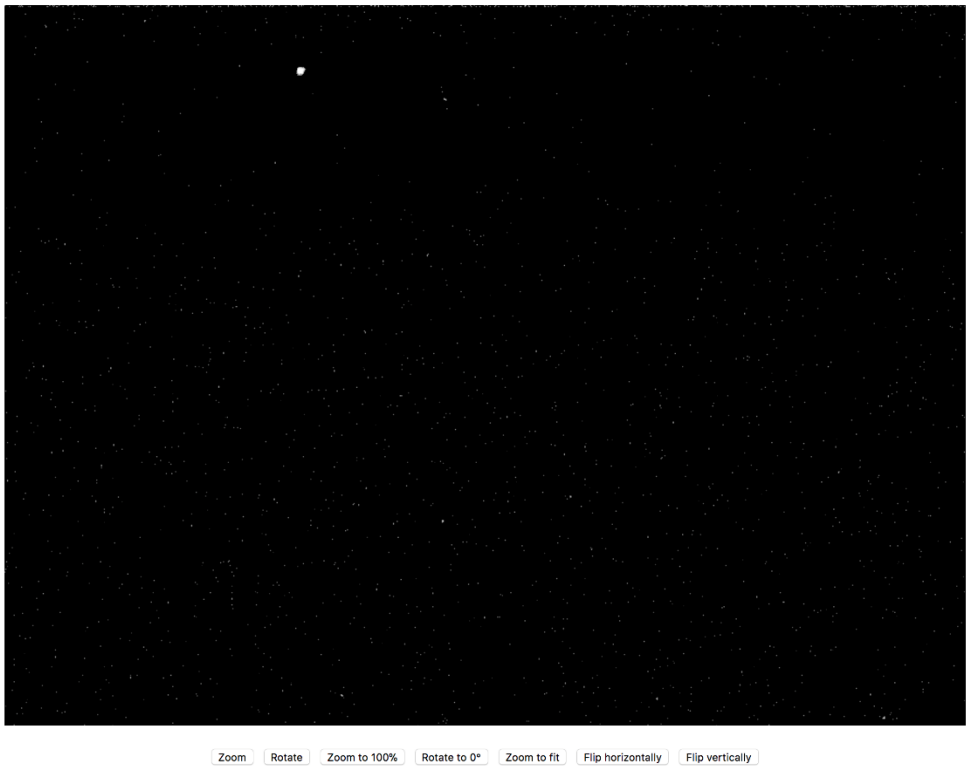

Figure 3. The canvas component displaying an image from a FITS file

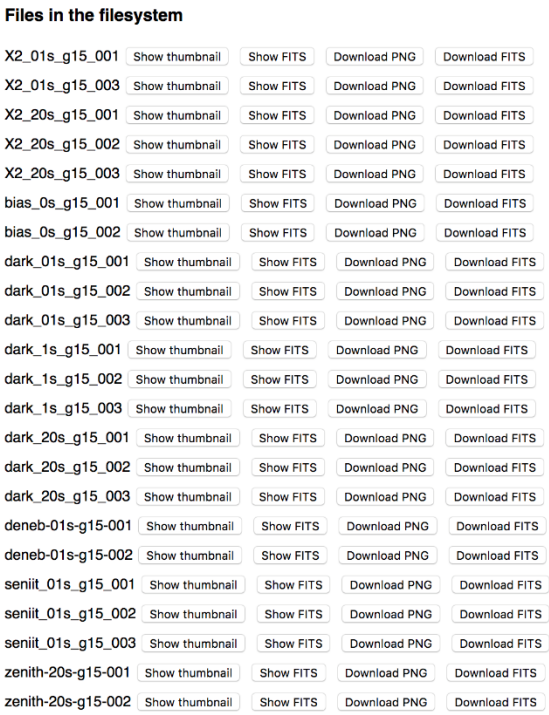

Figure 4. The gallery component displaying a list of images

5.5 Displaying Thumbnails on HTML Canvas

The core concept used in implementation of the canvas component is the HTML `canvas` element. The element was introduced in the HTML 5 specification [18]. It provides an interactive context and a JavaScript API for rendering of graphics on the fly. To display an image on `canvas`, the image source must be provided as one of the following data types:

- HTML `image` element;
- SVG `image` element;
- HTML `video` element;
- HTML `canvas` element [19].

Since the image files received from the satellite are transmitted as a bit array, the most suitable way to transfer this data onto a `canvas` is to create an HTML `image` element. It can be created with JavaScript using the `Image()` constructor:

```
const myImage = new Image(width, height);
myImage.src = 'http://webhost:port/image.jpg';
```

The image can now be placed onto a `canvas` in two steps. Firstly, we must fetch a reference to the two-dimensional rendering context of the `canvas` element. Secondly, we need to draw the image onto the `canvas` using the `drawImage()` method of the Canvas API.

```
const context = canvas.getContext('2d');
myImage.onload = () => {
       context.drawImage(myImage, 0, 0);
}
```

It is important to notice the usage of `onload` event handling in the previous code example. Since the loading of the image does not happen instantly, we must wait for the loading to complete before placing the element onto the `canvas`, otherwise the `drawImage()` method would instantly create an empty image on the `canvas`.

Using the method described above, it is possible to display the satellite image thumbnails saved using JPEG standard. It is only required to provide the correct address to the image that can be fetched from the back-end application.

Nevertheless, as discussed in the section 4.4, the files stored using the FITS standard have significantly more complex structure, and cannot be used as a direct data source to create an HTML `image` element.

5.6 Displaying FITS Images on HTML Canvas

One of the following solutions can be implemented to address the problem of showing the FITS images and the raw image data on an HTML `canvas` element.

1. Save the image data as a file in a common image format.
    a. Extract the required image data from the FITS file.
    b. Use the image data to create an image file in a conventional image format, such as Portable Network Graphics (PNG).
    c. Store the image file in a temporary data storage.
    d. Use the file as a data source for the `Image()` constructor and draw on the `canvas`.
2. Draw the raw image data pixel by pixel onto the `canvas`.
    a. Extract the required image data from the FITS file.
    b. Use the `context.createImageData()` method to create an array of pixel values in RGBA format.
    c. Place the `imageData` object contents onto the `canvas` using the two-dimensional context method `context.putImageData()`.
3. Use the experimental technology of the `canvas` API.
    a. Extract the required image data from the FITS file.
    b. Create an `ImageData` JavaScript object using the experimental `ImageData()` constructor.
    c. Place the `imageData` object contents onto the `canvas` using the two-dimensional context method `context.putImageData()`.

It is evident that all solutions require reading the contents of the FITS file and working with the raw image data. While it is possible to create a custom file-reading and data-parsing program, it is highly preferable to use an existing solution, as it allows to save significant amount of the development time and avoid accidental mistakes in the code. As mentioned in the section 4.4, many code libraries were written in different programming languages with an aim to simplify the process of working with the data stored using the FITS standard.

We inspected them to find the most suitable and commonly used solutions that could be integrated into our project.

As mentioned in the section 4.5, it is possible to work with the FITS files and the raw image data both in the front-end and in the back-end application. While server-side application has direct access to the filesystem, the client-side application would need to retrieve the file first. Unfortunately, since working with a file contents on the client side is not a common task, the FITS libraries written in JavaScript are few, and they do not have an active development community. Moreover, the existing JavaScript libraries, such as fitsjs [20], have not been updated for many years and cannot be used in a modern JavaScript project.

The fitsjs library relies on direct manipulation of the HTML Document Object Model (DOM) elements. Such approach produces errors in a React application, because React relies on the virtual DOM, which is a lightweight and detached from browser-specific implementation abstraction of the HTML DOM. The typical approach to using an external code library in React application is to import it as an isolated software module. The library code must be organised in a certain way to allow such use. Unfortunately, we did not discover any JavaScript libraries for FITS file operations that would meet this requirement.

Therefore, to read the data from a FITS file in the client-side application, it would be necessary to create a modern software library from scratch, or modify an existing open-source library such as fitsjs with the aim of "modernising" it. Unfortunately, this task could not be completed in the scope of the present work. We focused, instead, on analysing the FITS libraries written in other programming languages, and prototyping server-side solutions using such libraries. Since NASA's HEASARC FITSIO library is written in C programming language, and using C for web development is by far not a conventional approach [21], we inspected libraries written in other languages that would be more suitable for a microservice environment.

5.7 Development of the Go Prototype

While developing a new application in a microservice architecture, it is preferable to keep the number of programming languages involved as low as possible. This way it is not required for the present and future system developers to be proficient with too many languages and technologies. Following this principle, we chose to implement the first server-side application prototype with Go [22], as the language was previously used in the development of other server-side applications inside the ESTCube-2 Mission Control System.

Go, also known as Golang, is a programming language created by Google. Some experts compare Go to C and C++, pointing out excessive complexity of the latter languages and praising Go for simplicity in working on the same tasks [23]. The language is used for server-side computing at companies such as Google, Netflix, Dropbox, SoundCloud, Uber etc. [24]

The FITS I/O Libraries guide composed by the FITS Support Office at NASA/GSFC lists three FITS libraries written in Go [7]. For our project we used the fitsio library, as it is actively maintained and provides documentation. With help of the fitsio library it is possible to read the contents of the primary header-data unit of a FITS file and access the original data values with only few lines of code:

```
import (
   "os"
   fits "github.com/astrogo/fitsio"
)
func getFitsData(fname string) []int16 {
   r, err := os.Open(fname)
   f, err := fits.Open(r)
   hdu := f.HDU(0)
   hdr := hdu.Header()
   img := hdu.(fits.Image)
   nelmts := 1
   for _, axe := range hdr.Axes() {
      nelmts *= int(axe)
   }
   v := make([]int16, 0, nelmts)
   err := img.Read(&v)
   return v
}
```

The function `getFitsData()` in the code sample above reads a FITS file form the specified system path and returns the original image data values extracted from the primary HDU as an array of 16-bit integers. Unfortunately, we did not find a way to perform the scaling of data with help of this library, so the scaling would have to be implemented manually. Due to the lack of documentation, examples and experience with Go, we could not achieve server-side conversion of the image data to a conventional file format such as PNG in the scope of the present work. At this point several ESTCube-2 team members suggested to use a more common library, such as Astropy, which is implemented in Python. Since the Go prototype had several major issues, it was discard in favour of a new Python prototype.

5.8 Development of the Python Prototype

The Python programming language, together with a third-party NumPy library, is often used for numerical data processing and manipulation. It is, therefore, logical that the most popular software library for working with FITS file format is implemented in Python. As per definition, the Astropy Project is a community effort to develop a core package for astronomy using the Python programming language and improve usability, interoperability, and collaboration between astronomy Python packages [25]. As of March 2018, the open-source code repository consists of contributions of more than 200 individuals, producing 66 product releases.

The astropy.io.fits package provides functionality for working with FITS files. We used it in combination with Flask, a microframework for Python, to create a lightweight web server capable of reading and transmission of an image data.

```
from flask import Flask, jsonify, send_from_directory
from flask_cors import CORS
from astropy.io import fits
from astropy.visualization import make_lupton_rgb

app = Flask(__name__)
CORS(app)

@app.route('/api/v1/fits/<string:name>')
def fitsFile(name):
    data = fits.getdata('fits/' + name + '.fit').astype('float64')
    tempName = 'temp/' + name + '.png'
    make_lupton_rgb(data, data, data, filename = tempName))
    return jsonify(path = tempName)

@app.route('/temp/<path:path>')
def send_temp(path):
    return send_from_directory('temp', path)
```

In the code snippet presented above, we use the `fits.getdata()` function to extract the original image data from the primary HDU of the provided FITS file. A similar convenience function `fits.getheader()` can be used to extract the primary header information. Moreover, the extracted data is scaled automatically upon access, so the `data` array already contains the actual pixel values.

In the code example, we also make use of the Astropy visualisation function `make_lupton_rgb()` that allows us to automatically convert the actual pixel values into an array of 8-bit RGB values. This data can be used to plot the individual pixel values onto the HTML `canvas`. Calling this function also creates a PNG file in the temporary folder. This file, with adjusted white balance, can be displayed on a `canvas` or downloaded by user (see the requirement #8).

5.9 Canvas Image Operations

Once the satellite image is plotted onto the `canvas` element of the front-end application, it should be possible to perform a few image manipulations such as zooming in and out, panning, rotating, flipping and selecting a region of interest (requirements #3, #4, #5, #6). These actions can be performed in the client-side application using the HTML `canvas` API.

The functionality can be developed with JavaScript and the `canvas` API, using such two-dimensional context methods as `ctx.translate()` and `ctx.scale()`. This approach also requires implementation of a complex logical system for handling of the user's input (JavaScript mouse events `mousedown`, `mouseup`, `mousemove`) and calculating the relative positions of the canvas elements. It is, therefore, preferable to look for existing solutions for canvas manipulations, in order to simplify the task and rule out accidental code errors.

Following this logic, we used a popular `canvas` manipulation library that is compatible with React, React Konva. The library allows complex graphic operations on `canvas` and follows the React principle of component-based architecture [26]. The usage of the library provides valuable shortcuts compared to the direct API development. For example, to allow panning of the image (make it "draggable") with JavaScript and canvas API, it would be necessary to handle three mouse events and calculate the position, producing dozens of lines of code [27].

In React Konva, the interactive area consists of a combination of several `canvas` elements called `Stage`, `Layer` and `Image`.

```
<Stage
  ref={(stage) => { this.stage = stage }}
  scale={{x: this.state.stageScale.x, y: this.state.stageScale.y}}
  width={this.props.width}
  height={this.props.height}
>
  <Layer>
    <Image
      ref={(image) => { this.image = image }}
      scale={{x: this.state.imageScale.x, y: this.state.imageScale.y}}
      rotation={this.state.rotation}
      image={this.state.image}
      draggable
    />
  </Layer>
</Stage>
```

In the code example above, we create a `Stage` with a single `Layer` and a single `Image`, passing such parameters as image scale and rotation using the component properties and state. With React Konva panning of the image can be achieved by

applying a single argument to the `Image` object, `draggable`. It is still necessary to capture and handle the mouse events using JavaScript to update the component state and re-render the image upon change, but using the Konva library allows to significantly reduce the total code base.

6 Conclusions

In this work, we studied the possible approaches to creating a new software component for the ESTCube-2 Mission Control System. We gathered the functional requirements, inspected useful libraries, and described the implementation following the state of the art principles of software development. Taking into consideration the microservice architecture of the system, we analysed the technology capabilities, outcomes and drawbacks of delegating certain tasks to the client-side or the server-side execution. The summary of our findings follows:

1. Due to the lack of modern JavaScript libraries, there is no simple way to handle the Flexible Image Transport System files in the client-side application. A modern JavaScript library must be developed to perform the FITS file operations on the client side.
2. Due to the abundance of the FITS libraries developed in other programming languages, and the active involvement of the community with these libraries, handling of the FITS files in the server-side application does not require significant development effort.
3. Usage of the existing libraries, such as React Konva, can provide valuable shortcuts for the image manipulation tasks in the client-side application and significantly reduces the size of the code base.

Based on these findings, we developed several code prototypes and created the proof-of-concept software that implements the required functionality, following the approaches described in this paper. The screenshots of the React components are provided as Figure 3 and Figure 4 in the section 5.4. The links to the software source code can be found in the appendix I.

The work on the Mission Control System is ongoing and the presented solution is not a production-grade version of the software. A subset of required functionality remains to be developed, and the intention of the author is to continue the work on the software in collaboration with the ESTCube-2 team and to integrate the component fully into the system.

7 Bibliography

[1] SpaceWorks Enterprises, Inc. (SEI), "2014 Nano / Microsatellite Market Assessment," 2014. [Online]. Available: http://www.sei.aero/eng/papers/uploads/archive/SpaceWorks_Nano_Microsatellite_Market_Assessment_January_2014.pdf. [Accessed 28 02 2018].

[2] The CubeSat Program, Cal Poly SLO, "CubeSat Design Specification (CDS) REV 13," 20 2 2014. [Online]. Available: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf. [Accessed 28 02 2018].

[3] A. Slavinskis, M. Pajusalu, H. Kuuste, E. Ilbis, T. Eenmae, I. Sunter, K. Laizans, H. Ehrpais, P. Liias, E. Kulu, J. Viru, J. Kalde, U. Kvell, J. Kutt, K. Zalite, K. Kahn, S. Latt, J. Envall and P. Toivanen, "ESTCube-1 in-orbit experience and lessons learned," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 30, no. 8, pp. 12-22, 2015.

[4] H. Ehrpais, I. Sünter, E. Ilbis, J. Dalbins, I. Iakubivskyi, E. Kulu, I. Ploom, P. Janhunen, J. Sate, R. Trops and A. Slavinskis, "ESTCube-2 mission and satellite design," in *Small Satellites Systems and Services Symposium*, 2016.

[5] M. Mendez, The Missing Link. An Introduction to Web Development and Programming, Open SUNY Textbooks, 2014.

[6] I. Marsic, Software Engineering, New Brunswick, New Jersey: Rutgers University, 2012.

[7] NASA/GSFC, "The FITS Support Office," [Online]. Available: https://fits.gsfc.nasa.gov. [Accessed 28 02 2018].

[8] FITS Working Group, "FITS Standard Document, version 4.0," 22 07 2016. [Online]. Available: https://fits.gsfc.nasa.gov/standard40/fits_standard40aa.pdf. [Accessed 28 02 2018].

[9] The BPM Laboratory at the Koblenz University of Applied Sciences, “Status Quo Agile 2016/17,” 2017. [Online]. Available: http://www.status-quo-agile.net. [Accessed 28 02 2018].

[10] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, “Principles behind the Agile Manifesto,” 2001. [Online]. Available: http://agilemanifesto.org/principles.html. [Accessed 28 02 2018].

[11] Google, “Angular Documentation,” [Online]. Available: https://angular.io/docs. [Accessed 28 02 2018].

[12] Facebook, “React Documentation,” [Online]. Available: https://reactjs.org/docs. [Accessed 28 02 2018].

[13] J. Neuhaus, “Angular vs. React vs. Vue: A 2017 comparison,” 28 08 2017. [Online]. Available: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176. [Accessed 28 02 2018].

[14] D. Crockford, JavaScript: The Good Parts, Sebastopol: O’Reilly Media, Inc., 2008.

[15] M. Ostruszka, “It depends. The art of dependency management in JavaScript,” 06 02 2018. [Online]. Available: https://blog.softwaremill.com/it-depends-the-art-of-dependency-management-in-javascript-f1f9c3cde3f7. [Accessed 28 02 2018].

[16] Facebook, “Yarn Documentation,” [Online]. Available: https://yarnpkg.com/en/docs. [Accessed 28 February 2018].

[17] Facebook, “Create React App,” [Online]. Available: https://github.com/facebook/create-react-app. [Accessed 28 February 2018].

[18] W3C, “HTML5. A vocabulary and associated APIs for HTML and XHTML,” 28 October 2014. [Online]. Available: https://www.w3.org/TR/html52/semantics-scripting.html#the-canvas-element. [Accessed 28 February 2018].

[19] Mozilla, “Canvas API. Using images,” 27 November 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Using_images. [Accessed 2 March 2018].

[20] astrojs, “fitsjs,” [Online]. Available: https://github.com/astrojs/fitsjs. [Accessed 2 March 2018].

[21] L. Perkins, “Web development in C: crazy? Or crazy like a fox?,” 17 September 2013. [Online]. Available: https://medium.com/@lucperkins/web-development-in-c-crazy-or-crazy-like-a-fox-ff723209f8f5. [Accessed 3 March 2018].

[22] The Go Programming Language, “The Go Project,” [Online]. Available: https://golang.org/project/. [Accessed 3 March 2018].

[23] B. Eckel, “Calling Go from Python via JSON-RPC,” 27 August 2011. [Online]. Available: https://www.artima.com/weblogs/viewpost.jsp?thread=333589. [Accessed 3 March 2018].

[24] Wikipedia, “Go (programming language),” [Online]. Available: https://en.wikipedia.org/wiki/Go_(programming_language). [Accessed 3 March 2018].

[25] Astropy , “About The Astropy Project,” [Online]. Available: http://www.astropy.org/about.html. [Accessed 4 March 2018].

[26] A. Lavrenov, “React Konva,” [Online]. Available: https://github.com/lavrton/react-konva. [Accessed 4 March 2018].

[27] markE, “Make image drawn on canvas draggable with JavaScript,” [Online]. Available: https://stackoverflow.com/questions/15036386/make-image-drawn-on-canvas-draggable-with-javascript. [Accessed 4 March 2018].

Appendix

I. Source Code

The source code of the software developed in the course of this work is available online in the ESTCube-2 Bitbucket repositories.

- The front-end React prototype: https://bitbucket.estcube.eu/projects/ECIIMCS/repos/cam-fe
- The back-end Go prototype: https://bitbucket.estcube.eu/projects/ECIIMCS/repos/cam-be
- The back-end Python prototype: https://bitbucket.estcube.eu/projects/ECIIMCS/repos/cam-be2

Permission to access the code repositories is managed by the Estonian Student Satellite Foundation.

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Rodion Krjutškov**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Camera Component for the ESTCube-2 Mission Control System,

supervised by Dr. Andris Slavinskis and Alo Peets,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **09.03.2018**