



## Assignment of bachelor's thesis

<b>Title:</b>	Requests Status Overview Web Application based on Existing Task System
<b>Student:</b>	Lukáš Nymša
<b>Supervisor:</b>	Ing. David Šenkýř
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Information Systems and Management
<b>Department:</b>	Department of Software Engineering
<b>Validity:</b>	until the end of summer semester 2023/2024

### Instructions

This work is focused on companies using a task system to process client requests. The goal of this thesis is to design and develop a web application displaying the status of requests from the client's point of view.

Steps to follow:

1. Model an illustrative process for a company that:
  - a. processes requests from a client in an existing task system, and
  - b. communicates with a client only via e-mail messages.
2. Design a web application that should:
  - a. communicate with an existing task system,
  - b. display both unresolved and resolved requests of the verified client,
  - c. add comments to tasks representing unresolved requests.
3. Update the process from the first point following the proposed web application design in the second point.
4. Implement a prototype web application using .NET for YouTrack as a task system.
5. Evaluate and summarize the achieved results, including the economic-managerial evaluation.



Bachelor's thesis

**REQUESTS STATUS  
OVERVIEW WEB  
APPLICATION BASED  
ON EXISTING TASK  
SYSTEM**

**Lukáš Nymša**

Faculty of Information Technology  
Department of Software Engineering  
Supervisor: Ing. David Šenkýř  
11th May 2023

Czech Technical University in Prague

Faculty of Information Technology

© 2023 Lukáš Nymša. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

Citation of this thesis: Nymša Lukáš. *Requests Status Overview Web Application based on Existing Task System*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

# Contents

<b>Acknowledgments</b>	<b>vii</b>
<b>Declaration</b>	<b>viii</b>
<b>Abstract</b>	<b>ix</b>
<b>List of Abbreviations</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Analysis</b>	<b>3</b>
2.1 Task Systems . . . . .	3
2.1.1 Task Systems API . . . . .	3
2.1.2 YouTrack . . . . .	4
2.1.3 Asana . . . . .	4
2.1.4 Jira . . . . .	4
2.2 Common Processes of Communication Between Customers and Organizations . .	5
2.2.1 E-mail Communication Without Task System . . . . .	6
2.2.2 E-mail Communication Using Task System . . . . .	7
2.3 Web Application Analysis . . . . .	8
2.3.1 Requirements . . . . .	8
2.3.2 System States . . . . .	10
2.3.3 Analysis of Usage from a User's Point of View . . . . .	11
2.3.4 Analysis of Usage from an Employee's Point of View . . . . .	12
2.4 Analysis of Competitors . . . . .	12
<b>3 Design</b>	<b>13</b>
3.1 Application Technologies . . . . .	13
3.2 Back-end and Front-end Communication . . . . .	13
3.3 Back-end . . . . .	13
3.3.1 Application's Design Pattern . . . . .	14
3.3.2 Interface Mechanism for Different Task Systems . . . . .	15
3.3.3 YouTrack Connection . . . . .	15
3.3.4 Task State Mapping . . . . .	15
3.3.5 Configurable Connections and Settings . . . . .	15
3.3.6 Authentication . . . . .	16
3.4 Front-end . . . . .	16
3.4.1 Application's Design Pattern . . . . .	17
3.5 UI Design . . . . .	17
3.5.1 Welcome View . . . . .	18
3.5.2 List of Requests View . . . . .	18
3.5.3 Request's Detail View . . . . .	19

<b>4</b>	<b>Implementation</b>	<b>21</b>
4.1	Back-end	21
4.1.1	Directory Tree	21
4.1.2	Mediator Pattern	22
4.1.3	Commands	22
4.1.4	Controllers	22
4.1.5	Handlers	23
4.1.6	Services	23
4.1.7	Configuration Mapping	24
4.1.8	State Mapping	25
4.1.9	Authentication	26
4.1.10	Exceptions	27
4.1.11	Generated API Documentation	28
4.2	Front-end	29
4.2.1	State Management	29
4.2.2	Communication With Back-end	29
4.2.3	Error Handling	30
4.2.4	Authentication	30
4.2.5	Routing	32
4.2.6	Views	33
4.2.7	Dialogs	33
4.2.8	Light and Dark Mode Toggle	33
4.2.9	Responsive Design	34
4.2.10	Screenshots of the Implemented Web Application	35
<b>5</b>	<b>Testing</b>	<b>37</b>
5.1	Automated Testing	37
5.1.1	Unit Testing	37
5.1.2	Functional Testing	39
5.2	User Testing	40
5.2.1	Testing Process	40
5.2.2	Testing Results	41
5.2.3	Testing Summary	42
<b>6</b>	<b>Economic-Managerial Aspects</b>	<b>43</b>
6.1	Workflow Comparison	43
6.1.1	Evaluation of Steps to Produce	44
6.2	SWOT Analysis	44
6.3	Financing the Web Application	45
6.4	Future Outlook	46
<b>7</b>	<b>Summary</b>	<b>47</b>
	<b>Contents of the Attached Medium</b>	<b>51</b>

## List of Figures

2.1	E-mail communication without using task system . . . . .	6
2.2	E-mail communication using task system . . . . .	7
2.3	System states . . . . .	10
2.4	Communication from user's point of view . . . . .	11
2.5	Communication from employee's point of view . . . . .	12
3.1	UML class diagram of communication between task system services and handlers	15
3.2	Wireframe of welcome view . . . . .	18
3.3	Wireframe of list of requests view . . . . .	18
3.4	Wireframe of request's detail view . . . . .	19
4.1	State management Vuex [16] . . . . .	29
4.2	A list of requests on desktop (dark mode) . . . . .	35
4.3	Detail of request on desktop (light mode) . . . . .	35
4.4	A list of requests on mobile device on the left (light mode), detail of request on mobile device on the right (dark mode) . . . . .	36

## List of Tables

6.1	Cloud hosting services pricing comparison . . . . .	45
-----	---	----

## List of code listings

1	Example of command – <code>CreateTaskItemCommand</code> . . . . .	22
2	Example of endpoint – <code>POST /tasks</code> . . . . .	23
3	Example of handler – <code>ChangeTaskStateCommandHandler</code> . . . . .	23
4	Example of <code>YouTrackService</code> – <code>GetTaskAsync</code> . . . . .	24
5	Example of <code>Program.cs</code> – <code>MailSettings</code> Singleton . . . . .	25
6	<code>EnumExtensions</code> – <code>GetStringValue</code> . . . . .	25
7	Adding user to cache . . . . .	26
8	Searching for a user in cache . . . . .	26
9	Authorization filter method . . . . .	27

10	Example of exception handling middleware . . . . .	28
11	Dispatching an error in <i>actions.ts</i> . . . . .	30
12	Authentication data stored in Vuex . . . . .	30
13	Auto login function in <b>actions.ts</b> . . . . .	31
14	Vue routes . . . . .	32
15	Authentication guard . . . . .	32
16	Light and dark mode color setup . . . . .	33
17	Mock of <b>IProjectManagementService</b> with method setup . . . . .	38
18	Calling <b>createTaskItemCommandHandler</b> with command . . . . .	38
19	Verification of data and mocked method . . . . .	39



*I would like to thank my supervisor Ing. David Šenkýř for providing me with valuable advice throughout the whole process of creating the application and writing this thesis. I would also like to thank my family and partner for supporting me during my studies.*

## 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 accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on 11th May 2023

.....

## Abstract

This thesis aims at analyzing and developing a prototype web application (using .NET and Vue.js) that connects customers with organizations using task systems. The web application aims at allowing customers to create requests, view the status of created requests and communicate with organizations. At the same time, it allows organizations to communicate with customers only using task systems.

In the introduction of the thesis, the processes of the mentioned communication between the customer and the organization are illustrated with and without the use of the supporting application. In the conclusion, the economic-managerial aspects of the deployment and use of the implemented application are considered.

**Keywords** web application, task system, request status overview, .NET, Vue.js, API, YouTrack

## Abstrakt

Tato práce se zaměřuje na analýzu a implementaci prototypu webové aplikace (v technologiích .NET a Vue.js), která propojí zákazníky s organizacemi využívajícími systémy pro správu úkolů. Cílem této webové aplikace je umožnit uživatelům vytvořit požadavky, sledovat jejich stav a komunikovat s organizacemi. Zároveň umožňuje organizacím komunikovat se zákazníky pouze pomocí systémů pro správu úkolů.

V úvodu práce jsou ilustrovány procesy zmíněné komunikace zákazníka a organizace s využitím i bez využití podpůrné aplikace. Závěr práce zvažuje ekonomicko-manažerské aspekty nasazení a využití implementované aplikace.

**Klíčová slova** webová aplikace, systém pro správu úkolů, přehled stavu požadavků, .NET, Vue.js, API, YouTrack

## List of Abbreviations

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
ID	Identity
URL	Uniform Resource Locator
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
DTO	Data Transfer Object
SWOT	Strengths, Weaknesses, Opportunities, and Threats
CPU	Central Processing Unit
GB	Gigabyte
RAM	Random Access Memory
SSD	Solid-state Drive







### 2.1.2 YouTrack

YouTrack is a task management system developed by JetBrains<sup>4</sup>. YouTrack's API [2] is free to use and allows developers to create, modify, and delete tasks, projects and much more.

YouTrack meets all previously mentioned features in this Section 2.1. However, in YouTrack, tasks have different naming, they are called *issues*. Each YouTrack's issue has fields which contain some of the already mentioned features – project, assigned user and time tracking, yet they contain many more fields and YouTrack even allows the organization to create new custom fields. [3]

YouTrack can either be run on YouTrack's cloud server or YouTrack provides an installation package that allows installing it on the organization's own server. YouTrack offers different pricing plans and these plans differ for both platforms. For organizations of size of a maximum 10 people all features are for free on both platforms. If an organization requires more than 10 people in the task system, the pricing differs between these platforms. The cloud version is paid monthly or annually while the server version requires one-time license payment. [4]

### 2.1.3 Asana

Asana is a task management system developed by the same named company Asana, Inc<sup>5</sup>. Asana's API [5] is free and also allows developers to create, modify and delete tasks, projects and much more.

Asana likewise meets all mentioned features in this Section 2.1. Just like in YouTrack, data are stored in fields and Asana allows the organizations to create own custom fields.

Asana offers many pricing plans – *Basic*, *Premium* and *Business*. The *Basic* plan is free and allows up to 15 users and provides the organization with managing tasks, projects, messages and more features. The *Premium* plan can be paid monthly or annually and apart from the features from *Basic* plan it offers many more – unlimited users, timeline, workflow builder, custom fields and more. The *Business* plan can also be paid monthly or annually and on top of the features included in the previous plans it offers portfolios, workloads, time tracking and more. [6]

### 2.1.4 Jira

Jira is a task management system developed by Atlassian<sup>6</sup>. Jira offers free API that also allows developers to create, modify and delete tasks, projects and many more features.

Just like YouTrack and Asana, Jira meets all mentioned features in this Section 2.1 and data can also be stored in fields with the possibility to create custom ones.

Jira offers many plans *Free*, *Standard*, *Premium* and *Enterprise*. The *Free* plan can have a maximum of 10 users and is paid monthly or annually. This plan offers managing tasks, projects and more. The *Standard* plan can have up to 35 000 users and is also paid monthly or annually. It includes all features from the *Free* plan and also offers user roles, permissions, logs and more. The *Premium* plan contains all previously mentioned features, also offers a maximum of 35000 users and is paid monthly or annually. On top of that, it offers advanced road maps, archiving and more features. The *Enterprise* is paid annually and contains all previously mentioned features. It offers analytics, better support and can have unlimited users. The more users, the higher the price is.

---

<sup>4</sup><https://www.jetbrains.com>

<sup>5</sup><https://asana.com>

<sup>6</sup><https://www.atlassian.com>



## 2.2 Common Processes of Communication Between Customers and Organizations

Organizations could be made up by a few people or it could be a global organization of many employees. These global organizations usually have a department of people called help desk. Help desk is usually a group, but it could also be a single person, whose purpose is to help customers with a problem concerning their organization. [7]

In [7], there are described many ways how the customers can communicate with the help desk and how a help desk can deal with requests created by the customers. A list of examples of these communication ways follows:

- Phone – customers call the help desk and ask for real-time help.
- E-mail – customers contact the help desk via e-mail.
- Online forms – customers fill out an online form, for example on organization's website. They would usually provide e-mail or phone so the help desk can contact them in the future. This online form can then send these data to organization's database or to a task system where the help desk can deal with it.

In the following sections, some of these common communication processes are described. Firstly, the e-mail communication without using any task system is described in Section 2.2.1, then the e-mail communication with using task system is described in Section 2.2.2.

## 2.2.1 E-mail Communication Without Task System

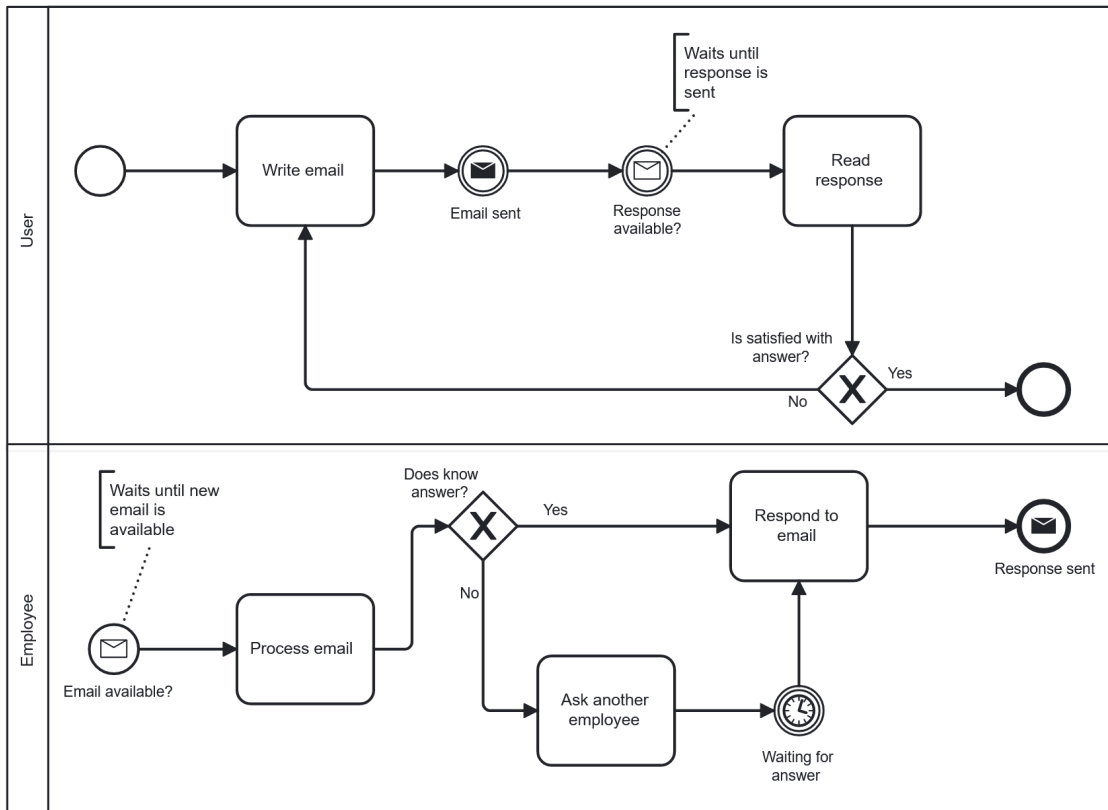
The firstly described process is by using e-mails only. Users write an e-mail with a request and then have to wait for a response from an organization. The employees do not always know how to answer the described request immediately and have to contact other employee with more knowledge about such a request via e-mail. This can take additional time. The employee then responds and the communication continues until a solution is provided and the user is satisfied with the answer. This process can be seen in Figure 2.1.

### 2.2.1.1 Advantages

- For issues that could be solved immediately, it can take less time since the communication is direct via e-mail.

### 2.2.1.2 Drawbacks

- User's e-mail can go unnoticed by the employees.
- Providing a response and whole solution for larger requests can take a longer time.



■ **Figure 2.1** E-mail communication without using task system

## 2.2.2 E-mail Communication Using Task System

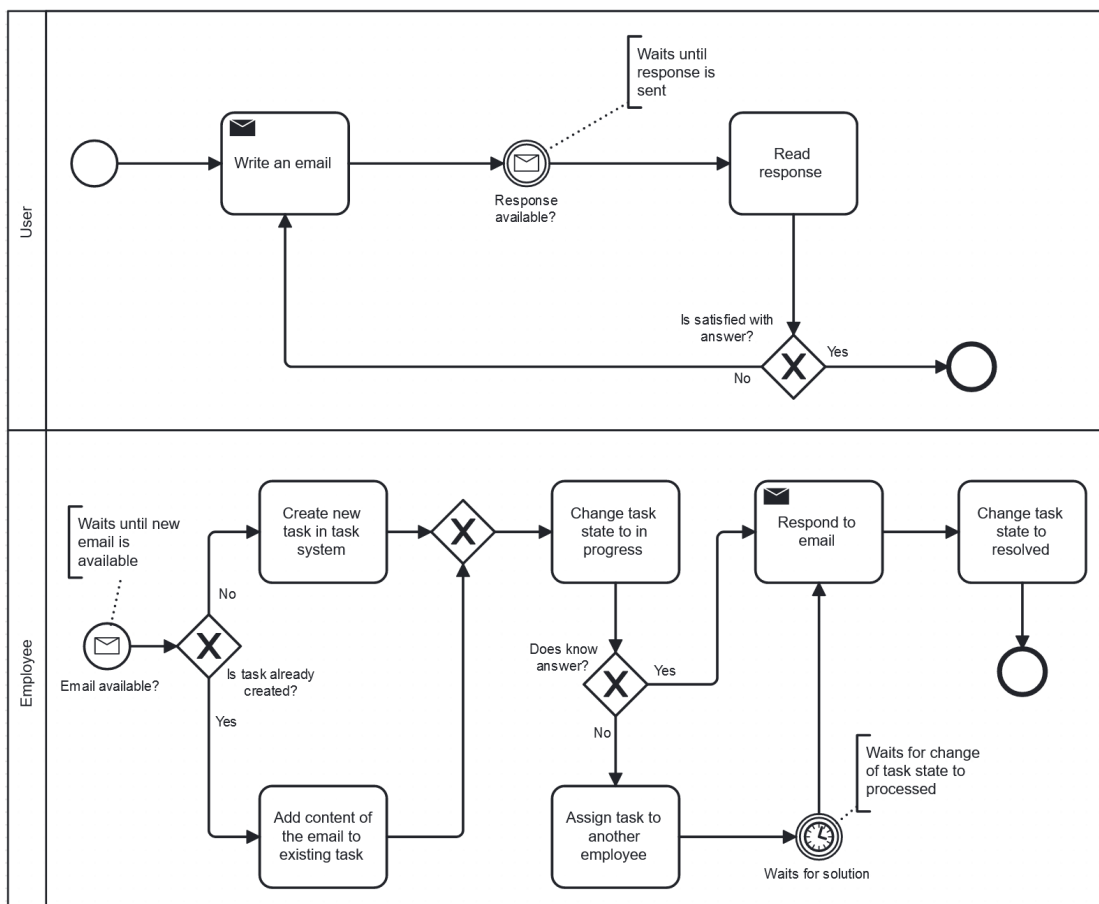
Organizations often use task systems which means they create new tasks for each user's request. A user sends an e-mail, the employee then creates a new task and assigns it to another responsible employee or themselves. The whole communication is done via e-mail, so the user has no idea whether the requests' resolution is in progress. Once the task gets solved in the system, the employee can then respond to the user. After that, the user and the employee communicate via e-mail until a solution is provided, while each question in the communication is also updated in the task system. This process can be seen in Figure 2.2.

### 2.2.2.1 Advantages

- Allows organizations to track all requests sent by users with its history and status.

### 2.2.2.2 Drawbacks

- User's e-mail can go unnoticed by the employees.
- Requires manual e-mail's content conversion into a new task in the existing task system.



■ **Figure 2.2** E-mail communication using task system

## 2.3 Web Application Analysis

From the analysis in the previous Section 2.2 (Common Processes of Communication), requirements are created for a web application that would simplify these processes. This web application would allow customers to create requests, see the statuses of these requests and send messages to the organization. All these requests would create new tasks in organization's task systems and the employees of the organization would be able to communicate with the customers only via the task system.

### 2.3.1 Requirements

In order to run a web application, it requires a back-end server for data. The server should also be connected to the task system for providing and sending data. The web application requires a user interface that communicates with the back-end server. The following sections describe functional and non-functional requirements of the web application.

#### 2.3.1.1 Functional Requirements

In this section, functional requirements are described. These requirements determine what the application should do and without them the application would not work as intended. [8]

- FR-1: Login
  - The user will be able to login using just his/her e-mail.
  - The user will provide his/her e-mail address on which a one-time code will be sent. The user will then provide that code to the application and become authenticated.
- FR-2: Authorization
  - The application will only allow access to tasks that are assigned to the logged in user.
  - Unauthenticated users will not be allowed to see anything other than the welcome page.
- FR-3: Access control
  - The application will only provide tasks to the user from projects that are allowed in the configuration of the application.
  - Newly created tasks will be added to the main project that is set in the configuration.
- FR-4: Task states
  - The application will work with the defined states of each task in the system so the users can see the progress of the request.
  - A list of states with explanation:
    - \* New: A user created a new task
    - \* Open: An employee noticed a newly created task
    - \* In Progress: The employee started solving the request
    - \* Retry: The employee sent the request back to the user because of incomplete information
    - \* Processed: The employee responded with an answer
    - \* Rejected: The user rejected the proposed answer
    - \* Resolved: The user is satisfied with the proposed answer
    - \* Reopened: The use corrected the information that were missing

- FR-5: Create a new task
  - The application will allow authenticated users to create a new task.
  - A task will contain a title and description.
  - The application will automatically pair the newly created task with the user's e-mail.
- FR-6: Add messages to an existing task
  - The user will be allowed to add messages to an existing task.
  - The messages will contain just a text.
- FR-7: View all existing tasks
  - The application will allow users to see all tasks which are assigned to them and list through them.
- FR-8: View an existing task
  - The application will allow users to see the specific task which is assigned to them.
  - Title and description will be provided as well as date of creation and all messages.

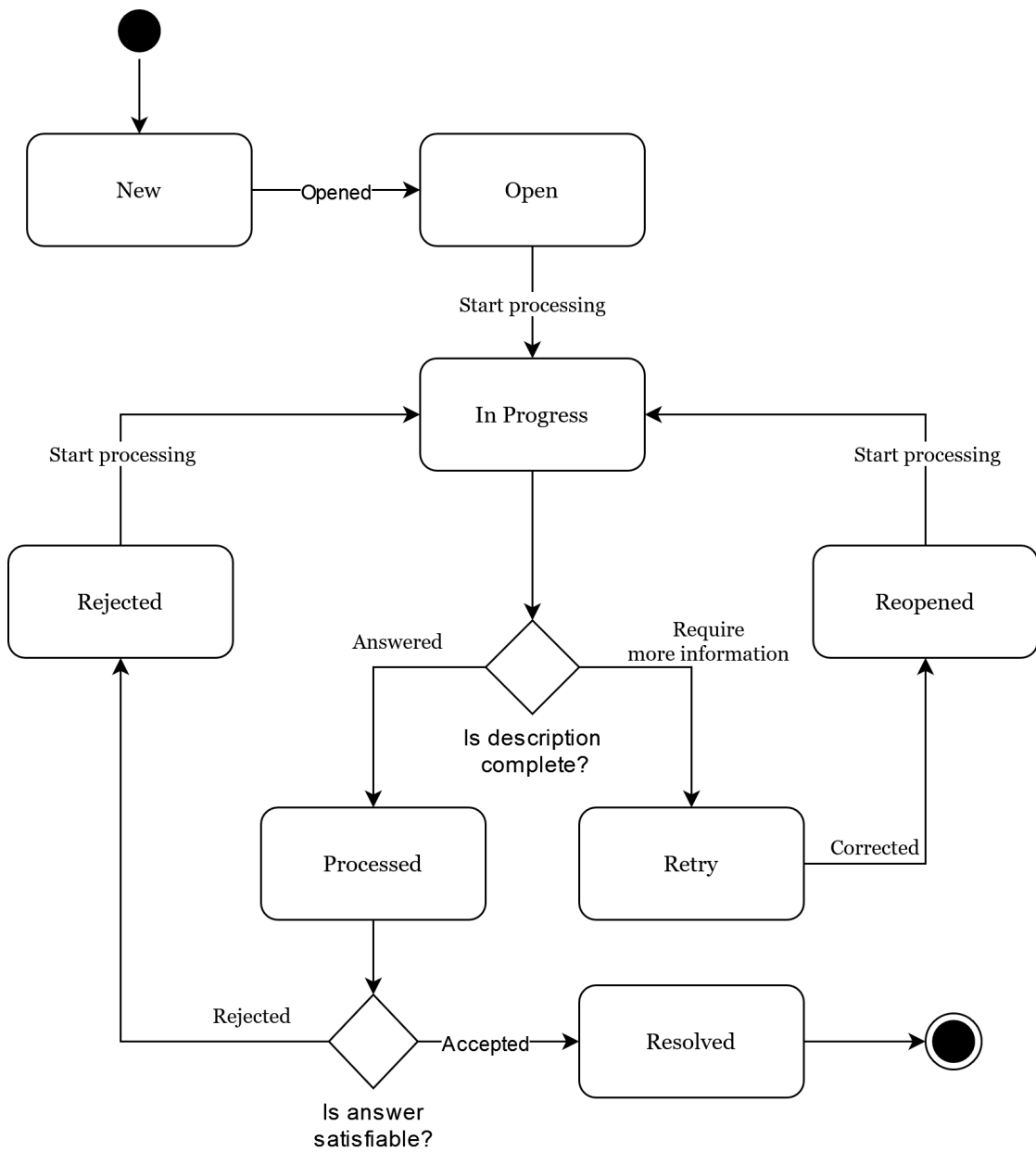
### 2.3.1.2 Non-functional Requirements

In this section, non-functional requirements are described. These requirements describe the quality of the web application, such as the extensibility or security, and the user's experience using this application. [8]

- NFR-1 A task system extensibility
  - The application should be extensible – it should allow developers to connect it to task systems that provide API.
- NFR-2 Encryption of communication
  - The application's communication should be encrypted.
- NFR-3 Device compatibility
  - The application should be available and usable on traditional web browsers as well as mobile devices.
- NFR-4 Open-source application
  - The application should be open-source meaning that the code is publicly available and any organization could use this code.

### 2.3.2 System States

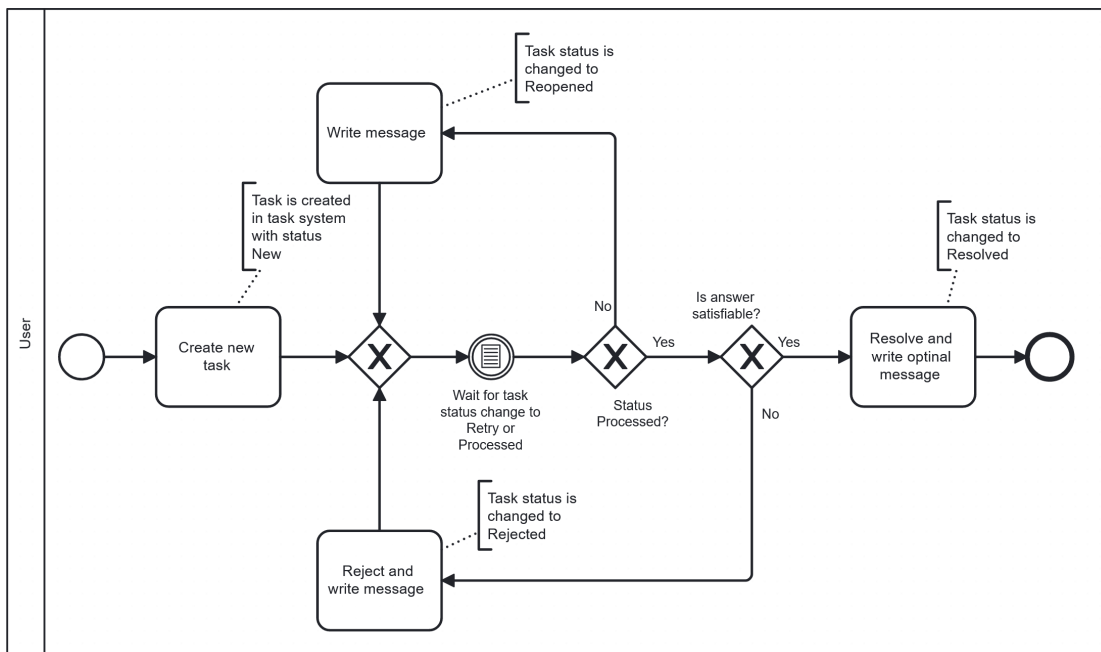
The web application should use states which would allow the users to see the progress of each created request. In the following Figure 2.3, there is a workflow of states defined in FR-4 2.3.1.1.



■ Figure 2.3 System states

### 2.3.3 Analysis of Usage from a User’s Point of View

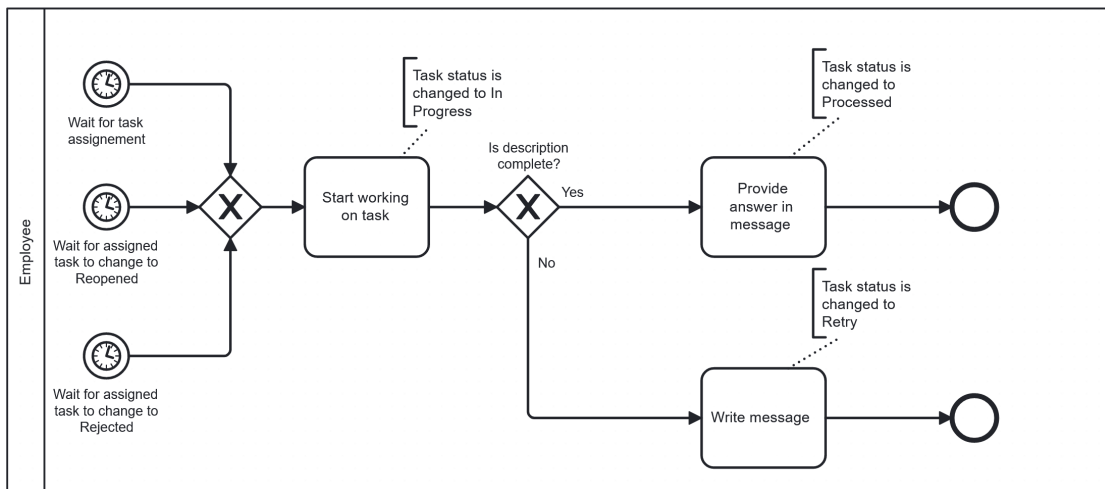
Users should communicate only via the web application. They should be able to create a new request that will be created in the task system. Then they have to wait until the state of created request is changed to either *Processed* or *Retry*. If the state is *Retry*, the users have to send additional information in the message because, according to the employee, the information given is not enough. However, if the state of the task is *Processed*, users then can proceed to either resolve the request or reject it. If the answer in the response message is sufficient, the task state is changed to *Resolved* state with an optional final message as an answer to the resolution. If the answer in the response message is not sufficient, the task state is changed to *Rejected* with a required message as an explanation of why it is rejected. The process then gets back to waiting until the state is changed to *Processed* or *Retry*. The process of user’s point of view is shown in the following Figure 2.4.



■ Figure 2.4 Communication from user’s point of view

### 2.3.4 Analysis of Usage from an Employee's Point of View

Employees should communicate with users only via an organization's task system. Once a new request is sent by a user, a new task is created in the task system and the task state is set to *New*. As soon as this task gets assigned to any employee, the state should be changed to *Open* and then the assigned employee can start working on the request. Once the employee resolves the request and provides an answer or solution, the state is changed to *Processed*. If the request is not complete and the employee needs more information, the state is changed to *Retry*. The employee starts working on request once the task is assigned to him/her or the already assigned task's state has been changed to *Reopened*. The process from the point of view of the assigned employee is shown in the following Figure 2.5.



■ Figure 2.5 Communication from employee's point of view

## 2.4 Analysis of Competitors

We were not able to find any other applications that would at least partially address the requirements mentioned in Section 2.3.1. As mentioned in Section 2.2, organizations can use online forms which customers can fill out and then the organizations process these requests. We do not consider this form of communication as a direct competition because the customers have no real-time feedback and can only rely on an e-mail or phone call back from the organization.

Since task systems often provide public APIs as mentioned in Section 2.1.1, organizations can implement their own applications that are not public and are used only inside the organization or they can build this communication directly into their own applications customers use.





## Chapter 3

# Design

The web application is designed according to the requirements from the previous Chapter 2 (Analysis). Since it is a web application, it is accessible to users through a web browser on both a computer and a mobile phone. It also allows the application to be updated and maintained without the users having to download or update the application themselves.

### 3.1 Application Technologies

We decided to split the web application into two separate applications, front-end and back-end. The front-end application is used for user interface and it is the application users interact within the web browser, while the back-end application is used to handle data – it processes data from the front-end and sends them to the task systems and vice versa.

### 3.2 Back-end and Front-end Communication

These two applications communicate with each other using HTTP requests. The front-end communicates with the back-end via its API. These requests contain JSON bodies which then get processed either in the back-end or front-end.

### 3.3 Back-end

For the back-end C# programming language was chosen as stated in Chapter 1 (Introduction) and in the assignment of this thesis. Since the application is for the web, we have chosen the ASP.NET framework, which is a free and open-source framework created and maintained by Microsoft. This framework is regularly updated to fix bugs and bring new functionalities as can be seen on the GitHub repository<sup>1</sup>.

The back-end does not need to store any data since it only works as a provider of data from a task system. For that reason, user's data, such as authorization tokens, are stored only in cache. In our selected approach, we do not use passwords for the user signup action. The application allows users to log in using only e-mail. First, a user provides an e-mail address, and the back-end then sends a one-time code to the provided e-mail address. The user can then authenticate using the one-time code.

---

<sup>1</sup><https://github.com/dotnet/aspnetcore>

The back-end application provides an API. It allows other applications to retrieve or send data to the application via HTTP requests. The application has many endpoints for authentication and data management. Each endpoint is listed and detailed in the following statements:

- `POST /users/request-code` – Requires a valid e-mail address, the application then sends a one-time code to the provided e-mail address.
- `POST /users/login` – Requires a valid e-mail address and provided one-time code. After validation, it provides a token to authorize the user with each request.
- `POST /users/logout` – Logs out the user in the application.
- `GET /tasks` – Provides all tasks that are assigned to the user.
- `GET /tasks/id` – Returns one task with provided ID.
- `POST /tasks` – Creates a new task with data provided in the body.
- `POST /tasks/id/comments` – Adds a new message to the task with the provided ID.
- `POST /tasks/id/approve` – Changes the state of task with provided ID to *Resolved*.
- `POST /tasks/id/reject` – Changes the state of task with provided ID to *Rejected* and requires message in the body.
- `POST /tasks/id/reopen` – Changes the state of task with provided ID to *Reopened*. This requires an explanatory message in the body.

### 3.3.1 Application’s Design Pattern

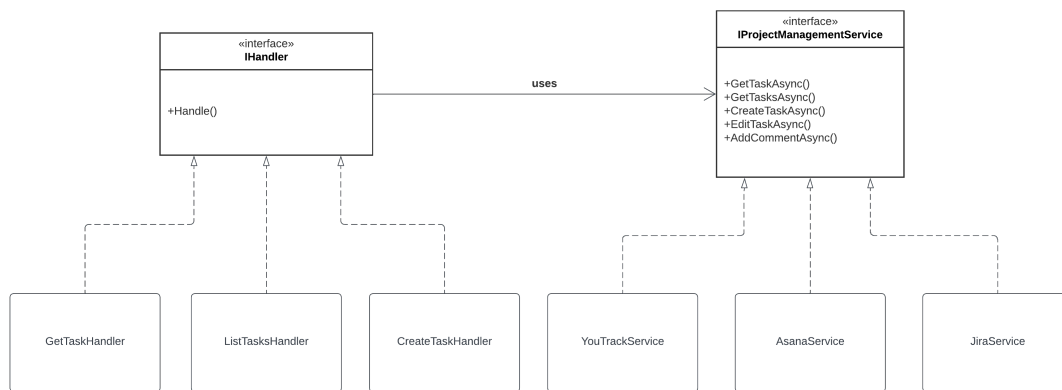
“*Design patterns are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.*” [9]

We have chosen to use the mediator behavioural pattern for the core of this web application. The purpose of this pattern is to have only one class *Mediator* that handles all dispatched commands. These commands are then redistributed to specific handlers that handle such commands. Each endpoint creates new commands that are dispatched to the mediator. The mediator then sends these commands to its handlers which proceed to do the desired action, such as sending a HTTP request to YouTrack’s API. This can be seen in the Figure 3.1.

In the application, we also decided to use singleton classes. As is stated in [10] “*Ensure a class only has one instance, and provide a global point of access to it.*” Singleton classes allow the application to access them anywhere in the application. These classes can be set at the build with static data and then accessed. Singletons are used in this web application for caching as will be further described in Section 3.3.6 (Authentication).

### 3.3.2 Interface Mechanism for Different Task Systems

The web application is designed to support many task systems that have available API. For that reason, the code design is unified and the application communicates with only one interface `IProjectManagementService` as is visible in Figure 3.1. For each task system, a new service can be implemented. This service must implement the interface `IProjectManagementService`. The interface has 5 different methods – to retrieve one or many tasks, create a new task, edit an existing task or add comments to an existing task. With this design, adding a service for another task system does not change the functionality of the web application. The application works the same and sends and retrieves data from another API only.



■ **Figure 3.1** UML class diagram of communication between task system services and handlers

### 3.3.3 YouTrack Connection

In our prototype, following the thesis assignment, we use YouTrack as a task system. As mentioned in 2.1.1 the application connects to YouTrack via its API. To connect the application to the API, an user in YouTrack needs to be created and the user's connection token is used by the web application. To recognize the author of each created task, a custom field needs to be set up in YouTrack which holds the user's e-mail. The setup of the token and custom field is done in a configuration which is further discussed in Section 3.3.5.

### 3.3.4 Task State Mapping

The web application uses 8 different states to show current progress. However, the used task system might have different states naming or the organization uses already its naming that is different from the state names of our implemented web application. For that reason, a mapping is used. Each state is mapped in the configuration to desired state name. How such configuration works is further described in the following Section 3.3.5. The usage of mapping also means that the visible task's state for the user might differ from the naming used in the task system.

### 3.3.5 Configurable Connections and Settings

The web application allows configuring connections and other settings. The application requires setting up a connection to YouTrack's API and own mail provider. It also needs to set up a custom field for user recognition as mentioned in Section 3.3.3 and mapping of states mentioned in Section 3.3.4.

The connection to YouTrack's API requires a URL to the server and the access token of user that will be marked as the creator of all the tasks created by users of this web application.

Connecting to a mail server or provider requires these details – host, port and authentication.

Setting up the custom field requires creating the field in YouTrack. Then, the name of such created field needs to be set in the configuration. Each created request then creates a task in the task system with the user's e-mail in this custom field.

Mapping of states only requires to define all state names in the configuration. First, the name of the state that is used in the application followed by the desired name that is used in the task system.

### 3.3.6 Authentication

The web application requires users to be logged in to get and send data. Since the web application is without a database, it uses a cache to store user data. Users can log in using just e-mail and a one-time code that is sent to provided e-mail. For that reason, the cache stores the user's e-mail, created one-time code, and token after successful login.

For data storing, a singleton class is used that allows the class to be accessible anywhere in the application. This class provides the application with methods to authenticate users and it also allows to validate each request.

## 3.4 Front-end

The front-end application is written in TypeScript which is a super-set of JavaScript with many more features. The advantages and disadvantages of TypeScript [11]:

- Advantages
  - Static typing – allows adding types to variables which means that once a variable is created with a certain type, the type cannot be changed and only values with the same type can be assigned to this variable.
  - Predictability – thanks to static typing, the code is much more predictable about what will happen and what can be the outcome of, for example, a function.
  - Error detection – TypeScript produces more notifications and exceptions directly in IDE which leads to fewer bugs, and more issues can be predictable. Using only JavaScript, many errors are found only during execution.
- Disadvantages
  - More code – Since TypeScript is a static language, it requires more code such as validation, invalid type prevention and much more.
  - Complexity – TypeScript needs to be compiled with every build which means it takes longer time, for example, in pipelines on GitHub.

Since the advantages outweigh the disadvantages, TypeScript is chosen for the front-end application, mainly because of typing. Other technologies that are used are HTML and CSS. HTML defines the structure of rendered content and CSS is used for design.

We decided to make the front-end application a single page. Single-page applications do not require reloading the whole page – when being redirected to the website, only one HTML file is being loaded and changed. This makes the application lot faster once loaded. [12]

Many frameworks simplify making such applications. The most known are React, Angular, and Vue.js. We have decided to use Vue.js for its simplicity. Vue.js is an open-source framework created in 2014. Vue.js is not as used or popular as React or Angular. According to Stackoverflow

2022 survey [13] which asked developers around the world what is their favourite language for web development, 19.9% responded with Vue.js, while 44.31% said React and 23.06% answered with Angular. Many well-known websites use Vue.js for the front-end, such as GitLab, Grammarly or Alibaba. One of the advantages that Vue.js has over other mentioned frameworks is a short learning curve – Vue.js does not require high prior experience with TypeScript, it is designed to be simple and fast without having to deal with complex setups. Another advantage is the size of the final application. The size of Vue.js applications is not huge, thus making it faster to load. [12]

### 3.4.1 Application's Design Pattern

The front-end application is split into views. Each view is then inserted into the page when accessed. Since web applications usually use the same parts of code (same visual parts) more than once, we use components. Components can be used for buttons, alerts and many more. These components are accessible in all views and can be used repeatedly.

Web applications need to store data, especially data about the user, such as for how long the user is still logged in. We decided to use a state management pattern that holds these data. It allows storing these data in one place, so these data do not have to be stored in each view or component. It not only improves code readability but also reduces risks of potential errors since the data are modified only at one place.

## 3.5 UI Design

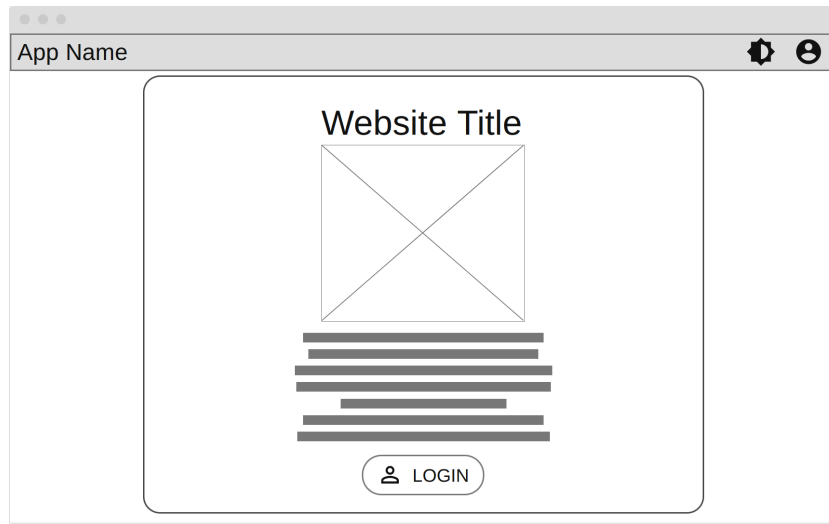
UI design is an important part of every web application that is going to be accessed by people with different computer experiences. It makes the first impression and it could discourage users from using it. That's why it is important to design it properly. Each view should be clear and consistent – similarly organized as the others. Since this web application is only for creating requests and subsequent communication, it should be as simple as possible and every step should be clear. The users of this web application vary, thus it is needed to be simple due to the user's experiences with computers differ.

The web application consists of 3 views. The welcome view, the list of requests view and lastly the request's detail view.

On top of that, a navigation bar is visible in all views. This navigation bar holds information about logged users. It shows the web application's title, login button and dark/light mode switch. The login button allows users to log in and if they are already logged in, it allows them to log out at any time. Each view is described in the following sections. Each section contains a wireframe – the blueprint of desired view design.

### 3.5.1 Welcome View

The welcome view consists of an explanation of the application, what steps the users have to do to resolve their issue and the login button. The welcome view is shown in the following Figure 3.2.

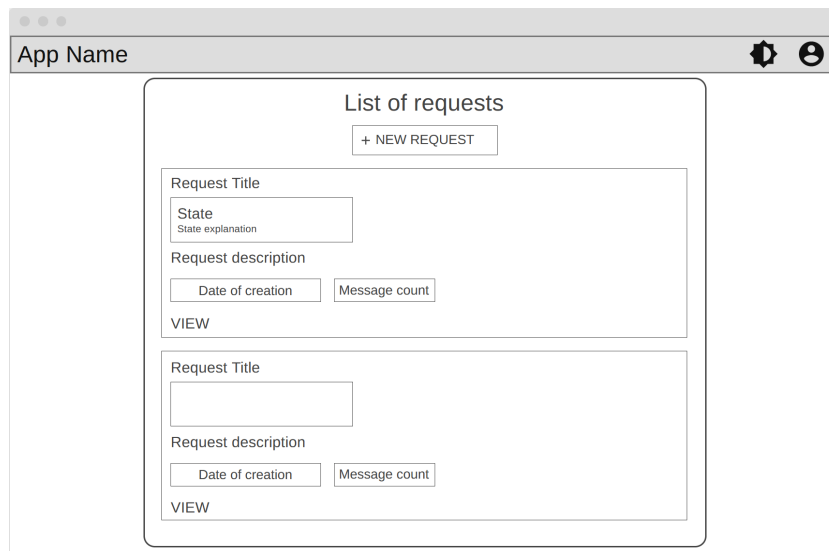


■ Figure 3.2 Wireframe of welcome view

### 3.5.2 List of Requests View

This view shows a list of all user requests and also allows users to create a new request.

Just below the view title, a button to create a new request is shown. Then there is a list with all user's requests. Each request contains a title, state with its description, request's description, date of creation and count of messages sent. According to these requirements, we designed this view as shown in the next Figure 3.3.



■ Figure 3.3 Wireframe of list of requests view

### 3.5.3 Request's Detail View

Detail view of a request displays all accessible information to the user: title, description, date of creation, current state and all messages. It also allows users to send new messages, and approve and reject solutions.

At first, the state with its description is shown. Then the date of creation and request description that was provided by the user. Below the description, a chat is displayed with all previous messages – date of creation and message's content. Users can also send new messages so below the chat there is input for messages with send button. The proposed design can be seen in the following Figure 3.4.



■ **Figure 3.4** Wireframe of request's detail view





# Implementation

This chapter contains all information about the implementation of the proposed design of the open-source web application. It is split into two sections: one is dedicated to the back-end development and one is dedicated to the front-end development.

## 4.1 Back-end

The application's back-end is implemented in ASP.NET 6 with the use of NuGet packages to simplify the application. These used packages are detailed further in this chapter.

### 4.1.1 Directory Tree

The back-end application is divided into multiple directories. Each directory contains classes with a different purpose. Some of these directories are further discussed in this chapter such as *Commands* in Section 4.1.3, *Controllers* in Section 4.1.4, *Exceptions* in Section 4.1.10, *Extensions* in Section 4.1.8, *Filters* in Section 4.1.9.2, *Handlers* in Section 4.1.5, *Services* in Section 4.1.6.

The *Config* directory contains singleton classes that are loaded and mapped from a configuration file. The *DTO* directory contains *Inputs* and *Queries* classes. These are used for request mapping in endpoints or mapping for the response of a request. The *Enums* directory contains enums that are used in this application, mainly for mapping data from task systems. The *Models* directory contains classes that represent some complex data, such as **Task**, **User** or **Comment**. For example, the **Task** class represents a task. To keep the code unified for all possible task systems connections, tasks from task systems are mapped to this class.

```
TaskSystem
├── Commands
├── Config
├── Controllers
├── DTO
├── Enums
├── Exceptions
├── Extensions
├── Filters
├── Handlers
├── Models
└── Services
```

## 4.1.2 Mediator Pattern

To implement the mediator pattern, we used an open-source NuGet package called MediatR<sup>1</sup>. It meets the description and functionality of the mediator pattern. It is popular since it has almost 10 000 stars on GitHub and it is regularly updated to meet with new updates in the ASP.NET framework. How is MediatR used in this application will be described in the following Sections 4.1.3 (Commands), 4.1.4 (Controllers), and 4.1.5 (Handlers).

## 4.1.3 Commands

To use MediatR we need to have commands that implement `IRequest` interface. Commands have the required class properties that are needed to handle such a command. It is a good practice to have these properties read-only (accessible via getters only) and provide them in the constructor.

### ■ Code listing 1 Example of command – `CreateTaskItemCommand`

```
public class CreateTaskItemCommand : IRequest<TaskItem>
{
    public string Name { get; }
    public string Description { get; }

    public CreateTaskItemCommand(string name, string description)
    {
        Name = name;
        Description = description;
    }
}
```

## 4.1.4 Controllers

Controllers are used to define actions. In our case, it exposes the API on the server. The application has 2 controllers, `TaskController` and `UserController`. Each method in controllers dispatches new commands that are then processed by the injected mediator instance of `IMediator`.

The `TaskController` is used for task handling – new requests from users, a listing of requests, sending messages and more. The `UserController` is used for authentication – requesting one-time code for the specific e-mail address, login and logout.

---

<sup>1</sup><https://github.com/jbogard/MediatR>

**Code listing 2** Example of endpoint – POST /tasks

```
[HttpPost]
public async Task<ActionResult<TaskItem>> Create([FromBody] TaskItemInput input)
{
    var command = new CreateTaskItemCommand
    (
        name: input.Name,
        description: input.Description
    );

    return Created(nameof(Get), await _mediator.Send(command));
}
```

## 4.1.5 Handlers

Once a command is dispatched using the MediatR mediator, it searches for handlers that implement `IRequestHandler<command, returned variable>`.

Each handler implements `IRequestHandler` and also has a `Handle` method which is executed after dispatching a command. The `Handle` method then executes wanted code, such as a call to a service (see the next Section 4.1.6).

**Code listing 3** Example of handler – `ChangeTaskStateCommandHandler`

```
public async Task<TaskItem> Handle(
    ChangeTaskStateCommand command,
    CancellationToken cancellationToken)
{
    var task = await _projectManagementService.GetTaskAsync(command.Id);

    if (task.TaskState != TaskStateEnum.Processed
        && task.TaskState != TaskStateEnum.Retry)
    {
        throw new UnprocessableDataException(
            $"Could not change state to {command.State} from {task.TaskState}");
    }

    return await _projectManagementService.EditTaskAsync(new TaskItem(command));
}
```

## 4.1.6 Services

Services are classes that could be described as parts of code that do the main part of the application – change the state of the application, send an e-mail or send requests to an external API.

The first described service is `MailService` which is only used to send e-mails with setting from the configuration file. It requires configuration data such as mail host and port, username and password. How is the configuration mapped is further discussed in Section 4.1.7 (Configuration Mapping).

According to the design proposition in Design chapter Section 3.3.2, an interface is implemented for different task systems. The interface is made up by 5 methods – `GetTasksAsync`, `CreateTaskAsync`, `GetTaskAsync`, `EditTaskAsync`, `AddCommentAsync`. Since it is an interface, each class that implements such an interface needs to have these 5 methods implemented. This thesis aimed to create an application that would connect to YouTrack. For that reason, `YouTrackService` is implemented and discussed in Section 4.1.6.1 (YoutrackService).

For handling authentication, `UserService` is used. It consists of methods to request a code, login, logout and currently logged-in users. It is mainly used by handlers that work with the authentication process.

#### 4.1.6.1 YouTrackService

As already mentioned, the `YouTrackService` implements `IProjectManagementService` with 5 required methods. This service is used for communication with the YouTrack API. YouTrack provides the NuGet package `YouTrackSharp` that allows developers to use this library to send and retrieve data from the YouTrack server without needing to implement any API request manually. To set up a YouTrack connection a server URL and permanent token are needed. Then with the method `Connect` from the library, the application connect to the YouTrackServer. `YouTrackSharp` also provides `IssueService` that allows developers to load and create issues (different naming for the task). We used this class to retrieve and send data to YouTrack as can be seen in the following Code listing `GetTaskAsync`.

##### ■ Code listing 4 Example of `YouTrackService` – `GetTaskAsync`

```
public async Task<TaskItem> GetTaskAsync(string id)
{
    var issue = await _issuesService.GetIssue(id);

    if (issue is null)
    {
        throw new EntityNotFoundException($"Unknown entity: {id}");
    }

    var authField = (List<string>)issue.GetField(UserFieldKey).Value;

    var projectId = GetProjectId(issue);

    if (!Projects.Contains(projectId)
        || !authField.Contains((await _userService.GetLoggedInUserAsync()).Email))
    {
        throw new EntityNotFoundException($"Unknown entity: {id}");
    }

    return YouTrackTaskItem.ConvertToTaskItem(
        issue, GetTaskStateFromIssue(issue), GetProjectId(issue));
}
```

#### 4.1.7 Configuration Mapping

The ASP.NET framework allows us to create an `appsettings.json` file that contains all configuration settings. This configuration can be accessed by `IConfiguration` by dependency injection.

To access the property from the `IConfiguration`, we can use `GetValue` method and then specify which variable to get like this:

```
configuration.GetValue<string>("Config:TaskSystem:YouTrack:Url")
```

This would retrieve a variable from objects `Config`, then `TaskSystem`, then `YouTrack` and finally the variable itself.

The ASP.NET framework uses `Program.cs` which is a code that runs during the build of the application. In this file, we can set up the whole application. Using that, we can set up a singleton class for `MailSettings` which loads settings from the configuration file and sets them to the class properties.

■ **Code listing 5** Example of `Program.cs` – `MailSettings` Singleton

```
builder.Services.Configure<MailSettings>(
    builder.Configuration.GetSection("Config:MailSettings"));
builder.Services.AddSingleton<MailSettings>();
```

## 4.1.8 State Mapping

As already discussed in Section 3.3.4, it is possible to map states with different naming to the states used in this application. These states are saved in `TaskStateEnum` which is an enum that holds these state names. For mapping, `EnumExtensions` class is implemented. It takes states from the configuration file and saves them into `EnumExtensions`' private variable `EnumValues` which holds at index the state naming used in the application and value is a naming used in the configuration file. It is also designed to be expandable for the future – the `EnumValues` is a dictionary within a dictionary for a case when more enums would need mapping to the task system's fields. The following Code listing 6 shows `GetStringValue` that is used when sending data to the task system's API. The function expects Enum as a parameter and returns a string that is used in the task system's field.

■ **Code listing 6** `EnumExtensions` – `GetStringValue`

```
public static string GetStringValue<T>(this T? value) where T : struct, Enum
{
    if (!value.HasValue)
    {
        return "";
    }

    var type = typeof(T);
    var name = Enum.GetName(type, value.Value);
    if (EnumValues.ContainsKey(type)
        && name is not null
        && EnumValues[type].ContainsKey(name))
    {
        return EnumValues[type][name];
    }
    return name ?? "";
}
```

## 4.1.9 Authentication

For authentication, there are 3 different endpoints implemented – `user/request-code`, `user/login`, and `user/logout`. Since the application does not have a database, a cache is used. For that, we created the `UserCache` model which contains the cached users.

### 4.1.9.1 Caching

To log in, the request-code endpoint must be first called which sends a code to an e-mail that needs to be then provided to log in. The request-code endpoint requires only e-mail while the login endpoint requires e-mail and the code sent in the request-code. For caching, we used `MemoryCache` from `Microsoft.Extensions.Caching`. The code for login and searching in cache is shown in the following Code listings 7 and 8.

#### ■ Code listing 7 Adding user to cache

```
public async Task<User> AddUserToTokenCacheAsync(User user)
{
    var cacheEntryOptions =
        new MemoryCacheEntryOptions().SetSlidingExpiration(
            TimeSpan.FromMinutes(TokenTtlMinutes));
    var createdUser = _tokenCache.Set(user.Token, user, cacheEntryOptions);
    await RemoveCodeFromCodeRequestCacheAsync(user.Email);
    return createdUser;
}
```

#### ■ Code listing 8 Searching for a user in cache

```
public async Task<User?> FindUserInTokenCacheAsync(string token)
{
    return await Task.Run(() =>
    {
        _tokenCache.TryGetValue(token, out User? user);
        return user;
    });
}
```

When adding a new user to the cache, sliding expiration is set. By default, it is set to 30 minutes but it can be modified in the configuration file. When the user is accessed in `FindUserInTokenCacheAsync`, the sliding expiration is reset and set again to 30 minutes. That means the user is logged in for 30 minutes since the last request.

The cache for requesting a code is the same but it does not use `_tokenCache` but rather uses `_codeRequestCache`. It is split for faster searching. For the one-time code cache, the index in the cache is an e-mail address. For the token cache, a token is the index since every request that is sent to API contains the token and not the user's email. For the request code, the index in the cache is e-mail while for the token cache token is the index since every request that is sent contains the token and not the user.

### 4.1.9.2 Authorization

To simplify validating if the user is authorized, we use filter. This filter can be set up in controllers, either for each endpoint or globally for all endpoints in the controller. `UserAuthorizationFilter` is implemented for the authorization. This filter is set up in `TaskController` with annotation `[ServiceFilter(typeof(UserAuthorizationFilter))]`.

With every HTTP request to any endpoint in this controller, the filter's method `OnActionExecutionAsync` is executed. This method checks the Authorization header and validates if the provided token is valid using `UserCache` as can be seen in Code listing 9. If the authorization fails, the `UnauthorizedAccess` exception is thrown.

#### ■ Code listing 9 Authorization filter method

```
public async Task OnActionExecutionAsync(
    ActionExecutingContext context,
    ActionExecutionDelegate next)
{
    if (context.HttpContext.Request.Headers.TryGetValues(
        "Authorization", out var tokenHeader)
        && tokenHeader.Any()
        && await _userCache.IsTokenValidAsync(tokenHeader[0]))
    {
        await next();
    }
    else
    {
        throw new UnauthorizedAccessException("Unauthorized");
    }
}
```

### 4.1.10 Exceptions

The application uses exceptions to signal errors. These exceptions can be thrown anywhere in the application and it is needed that these thrown exceptions are caught.

We use the `GlobalExceptionHandlerMiddleware` class which is a class that is invoked with an HTTP request. This means that if an exception has been thrown and not caught, the application does not fail but the `GlobalExceptionHandlerMiddleware` catches this exception in its `InvokeAsync` method. The `Invoke` method calls `next` which passes the request to the next component and if it fails, an exception is about to be caught depending on its type. If an exception occurs that is not expected, the global `Exception` catch block catches it. After catching these exceptions, a response is generated with the corresponding status code and context of the error. [14]

In the following Code listing 10, a part, since there are many exceptions handled, of the middleware's `InvokeAsync` method is shown.

**Code listing 10** Example of exception handling middleware

```
public async Task InvokeAsync(HttpContext context)
{
    try
    {
        await _next(context);
    }
    catch (UnauthorizedAccessException e)
    {
        await HandleExceptionAsync(
            context,
            e.Message,
            HttpStatusCode.Unauthorized
        );
    }
    catch (UnprocessableDataException e)
    {
        await HandleExceptionAsync(
            context,
            e.Message,
            HttpStatusCode.UnprocessableEntity
        );
    }

    // More catch blocks

    catch (Exception)
    {
        await HandleExceptionAsync(
            context,
            "Internal Server Error",
            HttpStatusCode.InternalServerError
        );
    }
}
```

### 4.1.11 Generated API Documentation

The ASP.NET framework offers documentation generation. We used Swagger<sup>2</sup> by using another NuGet package Swashbuckle<sup>3</sup>. In *Startup.cs*, we set up the configuration of this documentation generation using `builder.Services.AddSwaggerGen()`.

After starting up the application, a page on the URL defined in the configuration is created, the default URL is `<url>/swagger/index.html`. This page contains all endpoints the application exposes and also allows us to send HTTP requests to these endpoints via this page, mainly to test the functionality.

---

<sup>2</sup><https://swagger.io/>

<sup>3</sup><https://github.com/domaindrivendev/Swashbuckle.AspNetCore>



## 4.2 Front-end

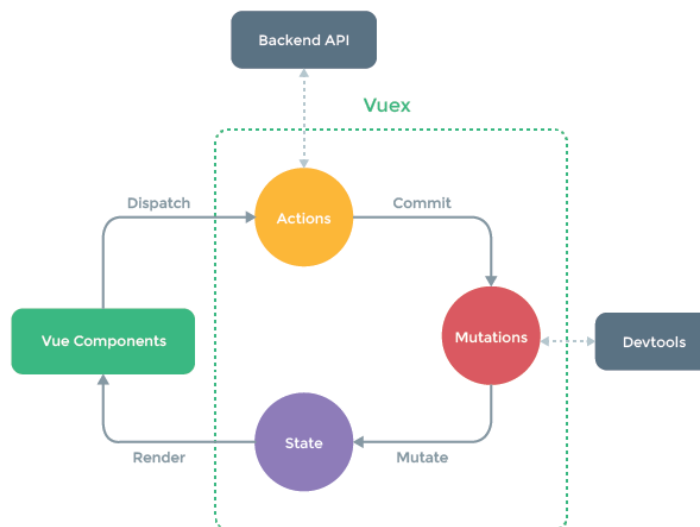
The front-end application is implemented in TypeScript using the Vue.js framework as well as HTML and CSS. To simplify the development, many libraries were used and they are discussed further in this section. To design the user interface, Vuetify [15] is used. Vuetify is based on Material Design<sup>4</sup> and offers many pre-made icons, components, and templates which simplify the development of the front-end application.

### 4.2.1 State Management

For state management, we used Vuex. “*Vuex is a state management pattern + library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion.*”[16]. Vuex has a state store which holds all data that are needed to be stored. To access these data from the store, getters are used to keep the same output throughout the application. These getters can be used in Vue components which render the page’s content. To modify data in the store, mutations are used, same as getters – to preserve the same behaviour throughout the application. The Vuex’s functionality can be seen in the Figure 4.1.

### 4.2.2 Communication With Back-end

To communicate with the back-end, the back-end’s API is used. Vuex provides *actions.ts* file, where all API HTTP requests are implemented using JavaScript’s `fetch`<sup>5</sup>. In these actions, data are loaded and then using mutations, committed to the store.



■ **Figure 4.1** State management Vuex [16]

<sup>4</sup><https://m3.material.io>

<sup>5</sup>[https://www.w3schools.com/jsref/api\\_fetch.asp](https://www.w3schools.com/jsref/api_fetch.asp)

### 4.2.3 Error Handling

In the web application, many errors can occur. Errors are dispatched via Vuex's *actions.ts* function `setError`. This function sets store's property `error` to a desired error text. By default, the property is empty and no error is being shown. Once the error is set to any text, pop-up appears. This error is set to be visible for 3 seconds and can be closed by user with close button.

Errors are usually dispatched when a HTTP request to the API fails, for example, the user is no longer authenticated or if there has been a problem with processing the request. The `setError` can be seen in the following Code listing 11.

■ **Code listing 11** Dispatching an error in *actions.ts*

```
async setError(context: any, payload: any) {
  store.commit('setError', {error: payload.error});
  setTimeout(() => {
    store.commit('unsetError');
  }, 3000)
},
```

### 4.2.4 Authentication

Authentication is done via simple forms. After opening the web application, the user is required to log in to access other parts of the application. To log out, the user menu is located in the top right corner where a logout button is shown.

All user data are saved in Vuex's store. The store contains information about the user and also has a property `error` which was discussed in Section 4.2.3 (Error Handling). These data stored in the Vuex are shown in the following Code listing 12.

■ **Code listing 12** Authentication data stored in Vuex

```
export interface State {
  email: string|null,
  token: string|null,
  error: string|null
}
```

To make sure that the user logs out after expiration time, a timer is used. Once the user logs in or sends a request, the timer is reset.

To keep the user logged in if he leaves the page and then comes back within the expiration time, Local Storage<sup>6</sup> is used. Once the user opens the application, the code to check if the data in the Local Storage are still valid is executed as can be seen in the Code listing 13. If these data are still valid, the user is then automatically authenticated.

---

<sup>6</sup><https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

The saved data to Local Storage are:

- *email* – User’s email that is set in login
- *token* – User’s token that is set in login.
- *tokenExpiration* – Timestamp of date time when the validity of the token ends. With every new HTTP request, the timestamp of the date time is extended by the *expirationTtl*.
- *expirationTtl* – Timestamp of the duration of the token’s validity that is being reset with every new HTTP request.

■ **Code listing 13** Auto login function in `actions.ts`

```
async autoLogin(context: any) {
  const email = localStorage.getItem('email');
  const token = localStorage.getItem('token');
  const tokenExpiration = localStorage.getItem('tokenExpiration');
  const expirationTtl = localStorage.getItem('expirationTtl');

  let expiresIn: number = 0;

  if (tokenExpiration) {
    expiresIn = +tokenExpiration - new Date().getTime();
  }

  if (expiresIn < 0 || expirationTtl === null) {
    await store.dispatch('logout');
    return;
  }

  await store.dispatch('resetTimer', {
    expiresIn: parseInt(expirationTtl)
  });

  if (email && token) {
    context.commit('setUser', {
      email: email,
      token: token
    })
  }
}
```

## 4.2.5 Routing

For routing, we used Vue Router. It allows us to split the content into different views. For each route, a view is inserted that contains all page's data. [17]

### ■ Code listing 14 Vue routes

```
const routes = [
  {
    path: '/',
    name: 'Home',
    component: () => Home,
    meta: { requiresUnauth: true }
  },
  {
    path: '/requests',
    name: 'TaskList',
    component: () => TaskList,
    meta: { requiresAuth: true },
    props: false
  },
  {
    path: '/requests/:id',
    name: 'TaskDetail',
    component: () => TaskDetail,
    meta: { requiresAuth: true },
    props: true
  }
]
```

To prevent unauthorized access, a guard is used. This guard is executed with every route redirect and it checks if a user is authenticated and if the target route requires authenticated user. The authentication for each route can be seen on the Code listing 14 with a flag `requiresAuth`. How such a guard is implemented is shown in the following Code listing 15.

### ■ Code listing 15 Authentication guard

```
router.beforeEach((
  to: RouteLocationNormalized,
  from: RouteLocationNormalized,
  next: NavigationGuardNext
) => {
  if (to.meta.requiresAuth && !store.getters.isAuthenticated) {
    next({ name: 'Home' });
  } else if (to.meta.requiresUnauth && store.getters.isAuthenticated) {
    next({ name: 'TaskList' });
  } else {
    next();
  }
})
```

## 4.2.6 Views

The web application consists of 3 views – Home, TaskList and TaskDetail. Each view contains 3 blocks – `<script setup lang="ts"></script>` for TypeScript code, `<template></template>` for Vue template from which an HTML is generated and finally `<style lang="sass"></style>` for CSS styling.

## 4.2.7 Dialogs

For pop-up boxes, we used dialogs. These dialogs are activated in views and are used for forms to send data. They are used in login and creating new requests.

## 4.2.8 Light and Dark Mode Toggle

We implemented switching between light and dark modes with a simple icon in the navigation bar. This is achieved by Vuetify's option to set default colors in *vuetify.ts*. These options then load data from the *vuetify.ts* and set colors from it. The colors used in this web application and the setup can be seen in the following Code listing 16.

### ■ Code listing 16 Light and dark mode color setup

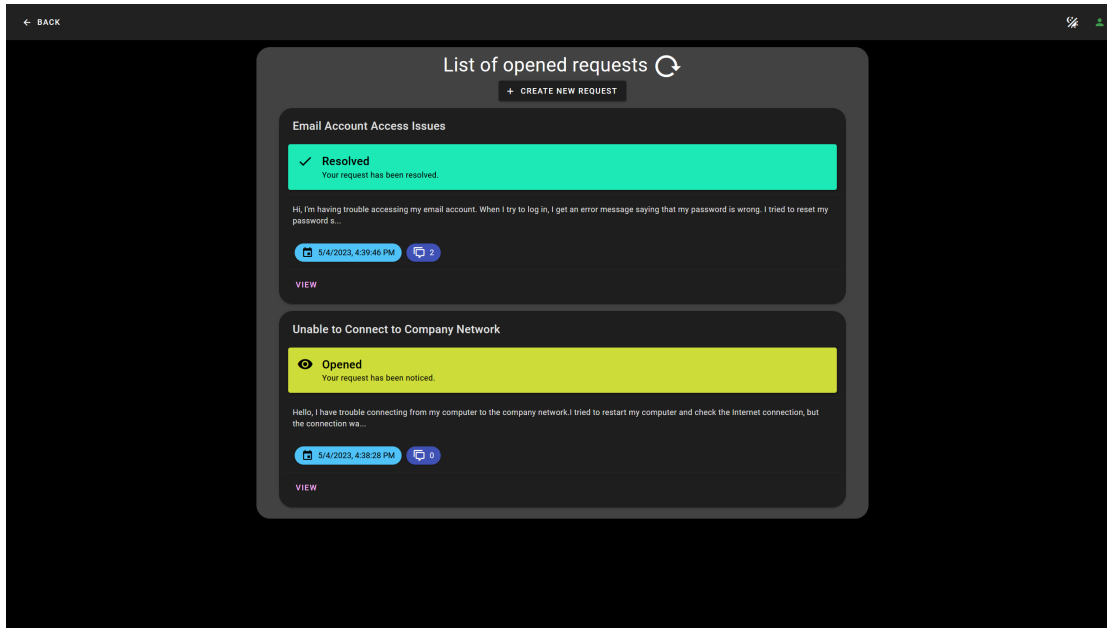
```
export default createVuetify({
  theme: {
    defaultTheme: 'dark',
    themes: {
      light: {
        colors: {
          surface: '#cccccc',
          primary: '#dadada',
          background: '#dadada',
          secondary_background: '#e8e8e8',
          bar: '#e8e8e8',
          text: '#000000',
          button: '#3F9E4A',
          error: '#cc0000'
        },
      },
      dark: {
        colors: {
          surface: '#1e1e1e',
          primary: '#1e1e1e',
          background: '#000000',
          secondary_background: '#424242',
          bar: '#212121',
          text: '#FFFFFF',
          button: '#3F9E4A',
          error: '#cc0000'
        }
      }
    }
  },
})
```

### 4.2.9 Responsive Design

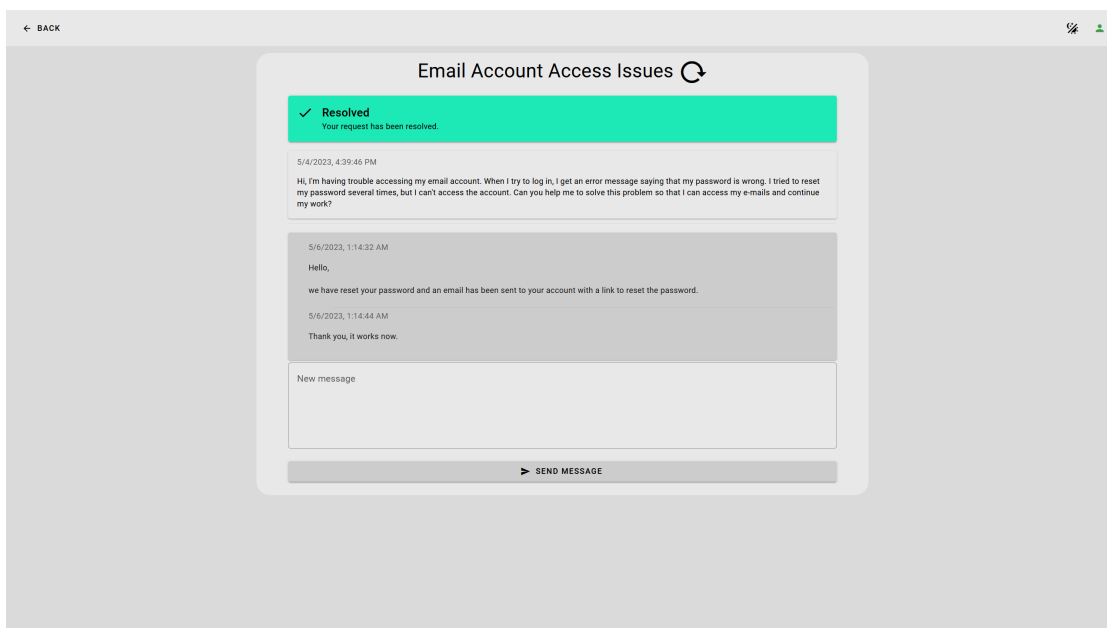
To achieve responsiveness between desktop and mobile environments, we used Vue's included function `useDisplay().mobile.value` which returns a boolean whether the used device is mobile or not. With this information, we then change the layout of the application in the views. The responsiveness can be seen in the next Section 4.2.10.

## 4.2.10 Screenshots of the Implemented Web Application

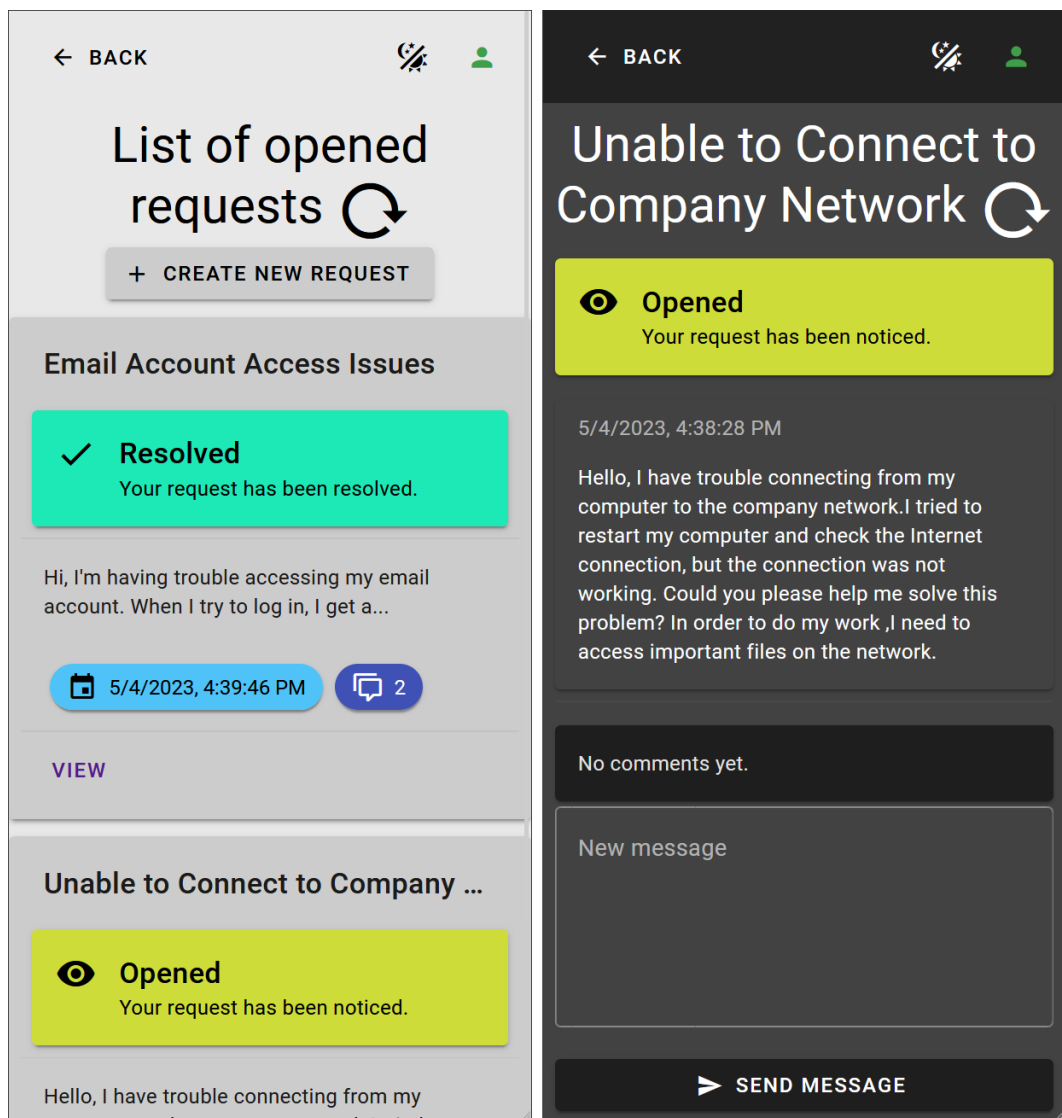
This section contains screenshots of the implemented prototype web application. Desktop version is shown in Figures 4.2 and 4.3. The mobile version is shown in Figure 4.4. For both desktop and mobile version, dark and light mode is shown.



■ Figure 4.2 A list of requests on desktop (dark mode)




■ Figure 4.3 Detail of request on desktop (light mode)



■ **Figure 4.4** A list of requests on mobile device on the left (light mode), detail of request on mobile device on the right (dark mode)





## Chapter 5

# Testing

This chapter is devoted to the testing of implemented prototype web application. First, automated testing is described. It involves unit testing and functional testing. Then this chapter is dedicated to user testing and the results of such testing.

### 5.1 Automated Testing

Automated testing provides developers to test an application without manual work. Automated testing can test applications as a whole or it can test parts of code, for example, classes or single methods. We did unit testing to test commands and their handlers and finally functional testing to test endpoints.

#### 5.1.1 Unit Testing

Unit testing is used to test units of code – one class at a time. Since we use the mediator design pattern, it allows us to test each handler.

For unit tests, we used the open-source xUnit <sup>1</sup> library from NuGet packages. It allows us to set up tests for each handler in different files. Each method that is supposed to be run during unit testing needs to be labelled with the `[Fact]` annotation. The xUnit library then notices this method and during unit testing, the method is executed.

To test handlers correctly, we need to use mocking. Mocking provides the developer with an opportunity to mock other classes – the classes implementing the main logic (workflow/behaviour) typically depend on other classes, for example, classes that provide data from some persistent store. However, our goal is to test the logic, and therefore mocking the supporting classes is useful. For mocking we used another library from NuGet packages called Moq <sup>2</sup>.

---

<sup>1</sup><https://xunit.net>

<sup>2</sup><https://github.com/moq/moq4>

### 5.1.1.1 Handler Unit Test Example

In this section, a unit test example for `CreateTaskItemCommandHandler` is shown. At first, a mock of `IProjectManagementService` is needed to set up methods that are called in this handler. This handler calls `CreateTaskAsync` method, so it needs to be set up with expected data that are going to be returned. Then the handler object can be created. This is shown in the following Code listing 17.

■ **Code listing 17** Mock of `IProjectManagementService` with method setup

```
var projectManagementService = new Mock<IProjectManagementService>();

projectManagementService.Setup(x =>
    x.CreateTaskAsync(It.IsAny<TaskItem>())).ReturnsAsync(
    (
        new TaskItem
        (
            "1",
            "Test name",
            "Description lorem ipsum.",
            TaskStateEnum.New,
            null,
            new List<CommentItem>()
        )
    );

var createTaskItemCommandHandler =
    new CreateTaskItemCommandHandler(projectManagementService.Object);
```

After that, the command is created and sent to the handler, so the handler then handles this command and uses the mocked method as can be seen in the following Code listing 18.

■ **Code listing 18** Calling `createTaskItemCommandHandler` with command

```
var command = new CreateTaskItemCommand(
    name: "Test name",
    description: "Description lorem ipsum."
);

var task = await createTaskItemCommandHandler.Handle(
    command, new CancellationToken()
);
```

The `task` variable contains a newly created task. Now the test checks if the data correspond with the provided data using assertions. Moq also provides the `Verify` method to verify what happened in the mocked method. In the following Code listing 19, we verify that the `CreateTaskAsync` has been only called once with provided data from the command.

■ **Code listing 19** Verification of data and mocked method

```
Assert.Equal(command.Description, task.Description);
Assert.Equal(command.Name, task.Name);

projectManagementService.Verify(x =>
    x.CreateTaskAsync(It.Is<TaskItem>(taskItem =>
        taskItem.Description == command.Description &&
        taskItem.Name == command.Name)
    ), Times.Once);
projectManagementService.VerifyNoOtherCalls();
```

The testing process of other handlers is similar to this. Every time the tests check that mocked methods have been called expected times and that the handler returns expected data.

## 5.1.2 Functional Testing

Functional testing is used for testing the complex functionality of the application. It does not test parts of code but individual endpoints sequentially. Since the web application requires a user to log in via e-mail and then pass the one-time code from the e-mail to the application, we created a testing user with a predefined token directly into the application's cache. This user is created only for testing and only if the testing user is set up in the configuration file.

We used Postman<sup>3</sup> for this API testing of endpoints. Postman offers a free plan and it allows users to send HTTP requests, make collections of HTTP requests and test each HTTP request individually. We created a collection with HTTP requests to test the process of creating a new request. Since the whole process of a request creation requires operations from the task system, it would require implementing endpoints for test scenarios, such as the status change, to act as an employee who changes and responds to created requests. For that reason, the testing collection only tests creating new requests, listing requests and trying to execute forbidden actions, such as changing state to *Rejected* even though the request is in state *New*.

The testing collection is detailed and explained in the following list of actions that are run sequentially:

- List requests – counts and saves the amount of already existing requests.
- Create new request – creates a new request with a randomly generated title and description. Saves these parameters.
- Get lastly created request – checks if the created request contains provided data.
- Reopen – tries to change the request's state to *Reopened*. It should fail since the request is in state *New* and return status code 422.
- Reject – tries to change the request's state to *Rejected*. It should also fail and return status code 422.

---

<sup>3</sup><https://www.postman.com>

- Approve – tries to change the request’s state to *Rejected*. It should fail like in the previous tests.
- Send a message to request – sends a new message to the created requests.
- Get created request – checks if the request contains a previously created message.
- List requests – counts created requests and compares the number with the previously saved amount.
- Logout – logs the testing user out.
- List requests – tries to list requests. Should be denied since the user is no longer authorized.
- Get created requests – tries to get created request. Should be also denied.

## 5.2 User Testing

User testing is a process in which testing subjects test a product, in this case, a web application. They have an objective that they need to achieve. The test leader usually provides the testing subjects instructions which they have to follow but the leader should not intervene with the subjects.

When deciding how many testing subjects are needed, we decided to follow Nielsen Norman Group’s article about their usability study [18]. Nielsen mentions in the article: “*Elaborate usability tests are a waste of resources. The best results come from testing no more than 5 users and running as many small tests as you can afford.*”. This statement comes from testing based on their previous studies. It shows that if we use more than 5 testing subjects, it is very likely that each additional testing subject is going to report the same things as the first 5 testing subjects. However, this depends on the scale of the tested projects. If, for example, there are multiple actions or objectives which the testing subject can achieve, then there should be 5 testing subjects for each action/objective. In our case, there is only one objective – create a request and get a solution. For that reason, we decided to perform the user testing with 5 testing subjects.

### 5.2.1 Testing Process

The testing was done on a local development computer since the application was not publicly available during the testing phase. Each testing subject has a different experience with computers, which gave us a view of the web application from different perspectives.

For testing purposes, a fictional organization was created and all the testing subjects were given the same objective – contact the fictional organization about a problem with their product. The problem was with the organization’s product – cloud photo storage. They were not able to upload and access photos on the cloud server.

At first, the testing subject was familiarized with the web application and what is its purpose. They were given a message from the organization that said they should contact them via a web application that serves the purpose of a help desk. Then each testing subject was given a sheet with steps to follow.

1. Login into the application.
2. Create a new request saying you have a problem with the organization’s product – an online photo editor. You cannot save your photos to the cloud and cannot access them.
3. You forgot to specify the application’s name since the organization offers many of them. Extend the request with information that you meant a product called PhotoX Cloud.

4. Wait for the test leader to provide a solution to your problem.
5. You find out that the solution is not helpful, and it still does not work. Let the organization know that you need another solution to your problem and wait for another solution.
6. You try the provided solution and find out that it works. Let the organization know.
7. Logout.

During the testing, we were acting as an employee of the fictional organization – answering and changing states of the created task. With each step the testing subject made, notes were made on whether the testing subject was sure about the process and what to do next or if the testing subject got stuck on a specific step and hesitated about what to do next. Each testing subject's experience with the web application and notes that were taken during testing are described in the following section.

## 5.2.2 Testing Results

### 5.2.2.1 Subject A

The first testing subject is a male, 26, and a web developer himself, so he is familiar with web applications. He understood the assignment and followed the steps until step number 5 without any problems. Once he had to reject the proposed solution, he hesitated about which button to press since there are two buttons: Resolve and Reject. He said there should be an explanation of what each button means and do. After that, he resolved the issue and logged out without any further problems.

### 5.2.2.2 Subject B

The second testing subject is a female, 23, and she is not familiar with IT at all and could be considered a usual user of computers and websites. She was able to log in and create new requests without any issues. Once she had to send a message that she forgot to specify the product's name as is stated in number 3, she struggled with finding out how to write a message. After a while, she managed to open the request's detail and sent the message without a problem. She then proceeded to resolve the issue and logged out successfully.

### 5.2.2.3 Subject C

The third testing subject is a female, 52, and she could be considered an occasional user of computers. She was able to log in and create a new request without a problem. Then she had to send the message that she forgot to mention the product's name, and she managed to do so. The problem occurred once she had to reject the proposed solution since it did not work as is stated in number 5, she was not sure which button to press, but in the end, she press the Reject button as she should. After that, she was able to resolve the issue and log out.

### 5.2.2.4 Subject D

The fourth testing subject is a male, 52, and he could be considered a casual user of computers. There were no issues with logging in, creating new requests and sending a message. Once he had to reject the proposed solution in number 5, he assumed that the green Resolve button would mean to send the message and the red Reject would close the whole request, so he resolved the issue even though he should have rejected it. This is an issue because the user is not able to change the state once the request is resolved. We changed the status to *Rejected* and he then managed to resolve the issue and log out.

### 5.2.2.5 Subject E

The fifth and last testing subject is a male, 23, and he could also be considered a casual user of computers. He was able to log in, create new requests and send messages without any issues. He struggled with deciding what button to press once he had to reject the proposed solution but managed to press the correct one – reject. Then he resolved the issue and logged out without any problems.

## 5.2.3 Testing Summary

Overall the testing was successful as there were no issues with the process of testing such as issues with the testing computer, the application or YouTrack.

The testing revealed some deficiencies in the application. Once the request is in the *Processed* state, users have to either resolve or reject the provided solution. This seems to be unclear as they all struggled with which button to press and one of the testing subjects even pressed the wrong button. Another imperfection of the application is opening request detail – it can be opened by clicking on the View button but one of the testing subjects could not find this button and tried to click everywhere but this button.

### 5.2.3.1 Possible Solutions to the Found Problems

To improve the problem of resolving and rejecting a request, few possible solutions are provided. One of these solutions would be a confirmation pop-up window that would specify what action is the user executing and what would happen. Another possible solution would be to remove these two buttons and replace them with a question “Are you satisfied with the provided solution?” with possible answers of yes and no. The user would then clearly decide whether the solution is sufficient or not. Then it would allow the user to write a message and send it.

The problem with viewing request detail could be improved by allowing the users to click anywhere on the whole request in the list. By clicking on it, it would open the request detail. That would mean the users would not have to only click on the view button but anywhere on the request resulting in a better user experience.

## Economic-Managerial Aspects

This chapter is devoted to economic-managerial evaluation. At first, a comparison between workflows of using and not using this web application is discussed, then SWOT analysis is detailed, then finally financing the application is evaluated, and finally the possible enhancements to the application are detailed.

### 6.1 Workflow Comparison

In this section, we compare two workflows of organization's employee in communication with customers. The first is using this implemented prototype web application and the second one is without using this web application and using e-mail communication with a task system only.

Workflow 1: The steps of an employee to solve a request using this prototype web application:

1. The employee 1 opens a newly created task in the task system.
2. The employee 1 changes task's state to Open.
3. The employee assigns the task to the employee 2 who can solve this issue.
4. The assigned employee 2 changes task's state to In Progress.
5. The assigned employee 2 provides a solution.
6. The assigned employee 2 changes task's state to Processed.

Workflow 2: The steps of an employee to solve a request without using this prototype web application:

1. New e-mail received, the employee 1 then creates a new task in the task system.
2. The employee 1 sets custom fields with the user's e-mail.
3. The employee 1 copies text from the e-mail to the task.
4. The employee assigns the task to the employee 2 who can solve this issue.
5. The assigned employee 2 changes task's state to Open.
6. The assigned employee 2 changes task's state to In Progress.

7. The assigned employee 2 provides a solution.
8. The assigned employee 2 changes task's state to Processed.
9. The assigned employee 2 assigns the task to the employee 1.
10. The employee 1 reads the solution from employee 2 and writes an e-mail to the customer.

If we assume that the request has been resolved after the provided solution, the workflow 1 is made by 6 steps, while the workflow 2 is made by 10 steps. The main advantage of workflow 1 is that it does not require reassigning the employees and once the assigned employee processes the task, the customer is aware of it, while in workflow 2, it requires the employee that has been communicating with the customer to write an e-mail to the customer.

If the customer rejects the provided solution, in workflow 1, steps 4 to 6 are repeated, that is 3 steps. However, in workflow 2, steps 3 to 10 are repeated apart from step 4 since the employee copies the e-mail to the already existing task. This means that employees in workflow 2 have to do 7 steps, which is 4 more steps than employees have to do in workflow 1.

### 6.1.1 Evaluation of Steps to Produce

To evaluate how much time would a organization save using this web application is a complex task. It depends on organization's size, meaning how many employees that deal with customers does the organization have. However, we are able to evaluate the amount of steps the employees have to do.

We were able to contact UA Support platform [19] and get data from them. The UA support platform matches described process of communication in 2.2 – they communicate with users via e-mail and for each request, they create new task in their task system. They provided how many requests they usually get weekly on their website. As we can see from the data, in the period from 9th September 2022 to 4th April 2023 almost 2200 requests have been managed. That is ~ 70 requests weekly.

For this organization, using the web application proposed in this thesis and assuming that the provided solution was not rejected, they would have to do weekly  $70 \cdot 6$  steps, that is 420 steps weekly. Without using this web application, they would have to do  $70 * 10$  steps weekly, that is 700 steps weekly. This means, without the web application, the employees have to do additional 280 steps weekly. For each request that gets rejected, the employee would have to do 4 more steps.

All steps that are in workflow 1 are also in workflow 2. That means this web application automates 4 steps. These steps are: creating a task, copying the content of an e-mail to the task, assigning back to the original employee, and sending an e-mail back to the customer.

In conclusion, the web application saves employees 4 steps to do with each request that has not been resolved. For requests that have been rejected once, the web application saves employees 8 steps. With each rejection, it takes 4 more steps without the web application.

## 6.2 SWOT Analysis

SWOT analysis is a technique that identifies strengths, weaknesses, opportunities and threats. The strengths and weaknesses describe the web application itself, such as what the application can and can not do. The opportunities and threats describe more the outside perception of the application, such as how much can the application be modified for different organizations or if the web application would be wanted by organizations. [20]

- Strengths:
  - Simplifies process of communication with customers.



- Could saves time – many of manual processes are automated.
- Weaknesses:
  - Requires a developer to setup the server and the application’s configuration.
  - Could discourage some people from contacting the organization – it might be too complex for them.
- Opportunities:
  - The web application is designed to be expandable. It, for example, allows connecting to other task systems apart other than YouTrack and also allows to read or modify more fields than the current ones.
  - Organizations could save money and time with usage of this web application.
- Threats:
  - Organizations might not be interested in using this web application.
  - Organizations might already have their own applications developed and do not want to use third-party options.

### 6.3 Financing the Web Application

To run the web application, hosting server is needed. Organizations could either use their own server or they can use a cloud hosting from other companies. There are many companies that offer cloud hosting available.

These cloud hosting services offer different plans depending on the power of the server. To choose how powerful the server should be depends on the expected traffic – how many customers would visit this website monthly. If we take for example the traffic from the previous Section 6.1, where the company receives ~ 280 requests monthly, a server with at least 2 core CPU and 3 GB RAM should be sufficient. These servers also have a maximum storage, in this case, storage is not that important, since the application does not have its own database and uses only caching. For that reason, any server with at least 10 GB SSD is enough.

We compared offers from different cloud hosting services (Hostinger<sup>1</sup>, Snackhost<sup>2</sup>, Upcloud<sup>3</sup>, Hetzner<sup>4</sup>, Vultr<sup>5</sup>) that would meet previously defined requirements. We outlined the CPU core count, RAM size, SSD capacity, and monthly fee as can be seen in the following table 6.1.

Cloud hosting service	CPU cores	RAM size	SSD capacity	Monthly fee
Hostinger	2 cores	3 GB	200 GB	16.99 €
Snackhost	2 cores	3 GB	10 GB	10.32 €
Upcloud	2 cores	4 GB	80 GB	26.00 €
Hetzner	2 cores	4 GB	40 GB	4.52 €
Vultr	2 cores	4 GB	25 GB	18.30 €

■ **Table 6.1** Cloud hosting services pricing comparison

<sup>1</sup><https://www.hostinger.com/cloud-hosting>

<sup>2</sup><https://www.snackhost.com/en/pricing-cloud-server/index.html@currency=eur.html>

<sup>3</sup><https://upcloud.com/pricing>

<sup>4</sup><https://www.hetzner.com/cloud>

<sup>5</sup><https://www.vultr.com/pricing>

## 6.4 Future Outlook

There are many possible enhancements to the implemented web application which could improve the overall user experience.

The web application could ask the user if he wants to extend the session so the application would not log the user out automatically. This could be done via a simple pop-up window.

Another possible enhancement could be the possibility to send files when creating a new request or sending a new message. Photos, for example, could be great addition since sometimes it is easier to explain a problem with a screenshot.

Sending e-mail notification if a state changes or new message shows in request could also be a great addition. This could be done in the back-end application with a command that runs in intervals. This command would save the state of the task to a cache and in each interval the command would compare it to the current state. If there are any changes, the application would send an e-mail to the assigned user to notify him.



- G2 – A web application that communicates with task systems was designed as two separate applications – front-end and back-end. The web application shows the current status and allows sending messages. This goal was achieved in Chapter 3 (Design).
- G3 – Analysis of the process using the proposed web application were fulfilled in Chapter 2 (Analysis).
- G4 – Prototype web application for the YouTrack task system was implemented successfully and described in Chapter 4 (Implementation). The web application is available on GitHub<sup>1</sup> (release 1.0.0).
- G5 – The economic-managerial aspects of the proposed web application was successfully evaluated in the Chapter 6 (Economic-managerial Aspects).

The web application has many possible modifications and enhancements that could be implemented to improve user experience. These enhancements were detailed in Section 6.4 (Future Outlook).

---

<sup>1</sup><https://github.com/lukasnymsa/connect-task-system>

# Bibliography

1. REITSMA, Tim. *What Is Task Management Software And How Can It Help You?* [online]. [visited on 2023-01-26]. Available from: <https://peoplemanagingpeople.com/articles/what-is-task-management-software>.
2. JETBRAINS. *YouTrack REST API* [online]. JetBrains s.r.o., 2023-02-24 [visited on 2023-02-26]. Available from: <https://www.jetbrains.com/help/youtrack/devportal/youtrack-rest-api.html>.
3. JETBRAINS. *Issues* [online]. 2023-04-28. [visited on 2023-04-29]. Available from: <https://www.jetbrains.com/help/youtrack/server/Issues.html>.
4. JETBRAINS. *Issues* [online]. [visited on 2023-04-29]. Available from: <https://www.jetbrains.com/youtrack/buy>.
5. ASANA. *Overview* [online]. [visited on 2023-04-29]. Available from: <https://developers.asana.com/docs>.
6. ASANA. *Pricing* [online]. [visited on 2023-04-29]. Available from: <https://asana.com/pricing>.
7. LOSHIN, Peter. *Help Desk* [online]. 2022-03. [visited on 2023-04-29]. Available from: <https://www.techtarget.com/searchcustomerexperience/definition/help-desk>.
8. CHITRASINGLA2001. *Functional vs Non Functional Requirements* [online]. 2022-12-02. [visited on 2023-05-01]. Available from: <https://www.geeksforgeeks.org/functional-vs-non-functional-requirements>.
9. REFACTORING.GURU. *What's a design pattern?* [online]. [visited on 2023-03-26]. Available from: <https://refactoring.guru/design-patterns/what-is-pattern>.
10. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. 37th ed. Addison-Wesley, 2009. ISBN 0-201-63361-2.
11. HOLMES, Joe. *TypeScript vs. JavaScript: 7 Key Differences* [online]. 2023-02-04. [visited on 2023-04-29]. Available from: <https://www.sanity.io/typescript-guide/typescript-vs-javascript>.
12. JOSHI, Mohit. *Angular vs React vs Vue: Core Differences* [online]. 2022-12-23. [visited on 2023-03-24]. Available from: <https://www.browserstack.com/guide/angular-vs-react-vs-vue>.
13. STACKOVERFLOW. *Web frameworks and technologies* [online]. 2023-01-26. [visited on 2023-03-27]. Available from: <https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe-prof>.

14. ANDERSON, Rick; SMITH, Steve. *ASP.NET Core Middleware* [online]. 2023-05-03. [visited on 2023-05-03]. Available from: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0>.
15. VUETIFY. *Vue Component Framework* [online]. [visited on 2023-04-26]. Available from: <https://vuetifyjs.com/en/>.
16. YOU, Evan. *What is Vuex?* [online]. 2023-09-03. [visited on 2023-04-26]. Available from: <https://vuex.vuejs.org>.
17. YOU, Evan. *The official Router for Vue.js* [online]. 2022-10-24. [visited on 2023-04-26]. Available from: <https://router.vuejs.org/https://router.vuejs.org>.
18. NIELSEN, Jakob. *Why You Only Need to Test with 5 Users* [online]. 2000-03-18. [visited on 2023-04-28]. Available from: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users>.
19. LINKING HELP; AGILAWYER; HOLUBOVÁ ADVOKÁTI; COPS STUDIO. *ABOUT THE PROJECT* [online]. 2023-04-04. [visited on 2023-04-26]. Available from: <https://www.ua.support/project-history/>.
20. GUARANA. *The Necessities of Running a SWOT Analysis for your App Idea* [online]. 2019-08-15. [visited on 2023-04-29]. Available from: <https://www.guarana-technologies.com/app-development/swot-analysis>.

# Contents of the Attached Medium

	readme.txt .....	brief description of contents of the attached medium
	src	
	impl .....	source code of the implemented prototype web application
	thesis .....	source code of the thesis in $\LaTeX$ format
	text	
	thesis.pdf .....	text of the thesis in PDF format