

**Bachelor Project**



**Czech  
Technical  
University  
in Prague**

**F3**

**Faculty of Electrical Engineering  
Department of Computer Science**

## **Integrated user interface for effective utilization of multiple project management tools**

**Anton Striapan**

**Supervisor: RNDr. Ladislav Serédi  
May 2021**



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Striapan** Jméno: **Anton** Osobní číslo: **478877**  
Fakulta/ústav: **Fakulta elektrotechnická**  
Zadávající katedra/ústav: **Katedra počítačů**  
Studijní program: **Softwarové inženýrství a technologie**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Integrované uživatelské rozhraní pro efektivní práci s více nástroji ke správě projektů**

Název bakalářské práce anglicky:

**Integrated user interface for effective utilization of multiple project management tools**

Pokyny pro vypracování:

Identifikujte možné scénáře paralelního způsobu práce s více nástroji pro zprávu projektů (project manager software, PM), podle Vaší volby, například: Jira, Tello a Asana. Najděte možné směry zvýšení efektivity činnosti. Diskutujte možnosti snižování času potřebného k přepínání mezi jednotlivými PM během práce. Navrhněte architekturu, jež umožňuje kombinovat jednotlivé PM a efektivně provádět běžné operace s nimi. Prozkoumejte stávající REST API jednotlivých PM, a na jeho základě vytvořte webovou front-end aplikaci (GUI) pro společnou správu projektů. Budou-li pro navrženou architekturu potřebné, implementujte serverové komponenty, případně vazbu na databázi. Navrhněte vhodný scénář použití Vaši aplikaci uživatelem z typického korporátního prostředí. Proveďte uživatelské testování, výsledky porovnejte a diskutujte výhody a nevýhody vašeho řešení oproti použití oddělených PM.

Seznam doporučené literatury:

1. Angular project website [online]. ©2010-2020 Google [27.12.2020]. Available at: <https://angular.io/>
2. Trello Developer Guides [online]. ©2020 Atlassian [15.12.2020]. Available at: <https://developer.atlassian.com/cloud/trello/rest/api-group-actions/>
3. API Introduction [online]. ©2020 Atlassian [15.12.2020]. Available at: <https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/>
4. Authorizing With Trello's REST API [online]. ©2020 Atlassian [15.12.2020]. Available at: <https://developer.atlassian.com/cloud/trello/guides/rest-api/authorization/>
5. Jira Server Developer Guides [online]. ©2020 Atlassian [17.11.2020]. Available at: <https://developer.atlassian.com/server/jira/platform/rest-apis/>
6. Asana Developers - Projects [online]. ©2020 Asana, Inc. [23.11.2020]. Available at: <https://developers.asana.com/docs/projects>
7. Ян Генрихович Малиевский, Руслан Иванович Баженов. Управление проектами в среде Trello [online]. ISSN 2414-4487 ©2015 Постулат [8.10.2020]. Available at: <http://www.e-postulat.ru/index.php/Postulat/article/view/3>
8. Jobin Kuruvilla. Jira Development Cookbook [online]. Packt Publishing Ltd, ©2016 [17.11.2020]. Available at: <https://books.google.sk/books?id=ioZcDgAAQBAJ>

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**RNDr. Ladislav Serédi, kabinet výuky informatiky FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2021**

Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

\_\_\_\_\_  
RNDr. Ladislav Serédi  
podpis vedoucí(ho) práce

\_\_\_\_\_  
podpis vedoucí(ho) ústavu/katedry

\_\_\_\_\_  
prof. Mgr. Petr Páta, Ph.D.  
podpis děkana(ky)

### III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

\_\_\_\_\_  
Datum převzetí zadání

\_\_\_\_\_  
Podpis studenta

## Acknowledgements

I'm very grateful to my mentor RNDr Ladislav Serédi for his big part in my writing assignment, for the useful comments which he gave me during the semester. Also, I would like to thank my supervisor from Siemens - Michal Rydlo for his help with the project and all other people who were involved in the work on my thesis.

## Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May , 2021

## Abstract

The goal of this bachelor thesis is to propose a way to increase the effectiveness of task managing in cases when individual tasks are stored and managed by multiple applications. This goal can be achieved by eliminating the time and effort spent on switching between different task managers. The task managing software selected for this thesis includes Asana, Jira and Trello. Their REST APIs are discussed in the Research section. The tool proposed in the Implementation part will allow combining tasks from these task managers in a single GUI tab enabling all common operations in one place, effectively eliminating the need to separately log in to each task manager. This tool will be implemented as a web application, with a front-end running in a browser, communicating with each of the task manager through their REST API. It will manage authorization to task managers as well – for this purpose, it will include a back-end and database component. After building a working prototype, user testing will be performed to show the effectiveness of the described approach. This work was supported by Siemens Czech Republic to increase the effectivity of usage of project managers and its internal processes.

**Keywords:** Trello, Jira, Asana, Task Managers, Project Managers, Time Tracking

**Supervisor:** RNDr. Ladislav Serédi  
Praha, Resslova 9, E-429 (vchod Karlovo náměstí 13)

## Abstrakt

Cílem této bakalářské práce je navrhnout způsob zvýšení efektivity správy úkolů v případech kdy jednotlivé úkoly jsou uloženy a spravovány v různých nástrojích. Toho může být dosaženo odstraněním prodlevy nastávající během přepínání mezi jednotlivými správci úkolů. Nástroje – správce úkolů - vybrané pro tuto práci jsou Asana, Jira a Trello. Jejich REST rozhraní je rozebrán v první části práce. Softwarový nástroj navržený v implementační části bude dovolovat kombinování úkolů od jednotlivých správců do jediné GUI komponenty bez nutnosti opakovaného přihlášení do jednotlivých správců úloh. Nástroj bude implementován jako webová aplikace, s částí front-end běžící v prohlížeči komunikující se správci úloh prostřednictvím jejich specifického REST rozhraní. Bude mít též na starosti autentifikaci uživatele směrem ke správcům – z toho důvodů bude nástroj disponovat též databázovou a back-end komponentou. Po implementaci funkčního prototypu proběhne uživatelské testování se zaměřením na použitelnost a efektivitu popsaného přístupu. Práce byla podporována společností Siemens Česká Republika jako součást jeho úsilí o zefektivnění použití správců projektů.

**Klíčová slova:** Trello, Jira, Asana, Správci úloh, Projektoví manažeři, Sledování času

**Překlad názvu:** Integrované uživatelské rozhraní pro efektivní práci s více nástroji ke správě projektů

# Contents

<b>1 Introduction</b>	<b>1</b>
<b>2 Done research</b>	<b>3</b>
2.1 REST API .....	3
2.1.1 Trello .....	3
2.1.2 Jira .....	4
2.1.3 Asana .....	4
2.2 Authorization in provided REST API from Trello, Jira, Asana .....	4
2.2.1 Trello .....	4
2.2.2 Jira .....	5
2.2.3 Asana .....	6
2.3 Fetching tasks from PM .....	7
2.4 Used technologies .....	7
2.5 Summary .....	8
<b>3 System architecture</b>	<b>11</b>
3.1 System overall diagram .....	11
3.2 Database UML diagram .....	12
3.3 Trello Authorization flow .....	13
3.4 Jira Authorization flow .....	13
3.5 Asana Authorization flow .....	15
<b>4 Implementation</b>	<b>17</b>
4.1 Authorization .....	17
4.2 Internal Authorization .....	17
4.2.1 Database .....	18
4.2.2 Front-end .....	18
4.2.3 Back end .....	19
4.3 Authorization in PM's .....	21
4.3.1 Database .....	21
4.3.2 Front-end .....	21
4.3.3 Back end .....	25
4.4 Fetching tasks from PM .....	27
4.4.1 Trello .....	27
4.4.2 Jira .....	28
4.4.3 Asana .....	29
4.5 Displaying Tasks in front-end ..	31
<b>5 Testing</b>	<b>35</b>
5.1 Artem Hurbych (CTU, FEE, OI 3rd year student) first version testing .....	35
<b>6 Conclusion</b>	<b>37</b>
<b>A Application setup instructions</b>	<b>39</b>
A.1 How to run and serve application	40
<b>B List of Abbreviations</b>	<b>41</b>
<b>C Bibliography</b>	<b>43</b>

## Figures

## Tables

2.1 Trello authorization flow . . . . .	5
2.2 Jira authorization flow . . . . .	6
2.3 Asana authorization flow . . . . .	7
2.4 Angular Model . . . . .	8
3.1 System overall diagram . . . . .	11
3.2 UML class diagram . . . . .	12
3.3 Trello Authorization flow . . . . .	13
3.4 Jira Authorization flow . . . . .	14
3.5 Asana Authorization flow . . . . .	15
4.1 Registration form from frontend	18
4.2 Login form from frontend . . . . .	19
4.3 UI for account PM connetions page . . . . .	22
4.4 Trello granting access page . . . . .	22
4.5 Jira access granting page . . . . .	23
4.6 Jira form for code . . . . .	24
4.7 Asana access granting page . . . . .	24
4.8 Filled table with tasks from Trello/Jira/Asana . . . . .	31
4.9 Opened task with description . . . . .	32
4.10 Opened task with filled comments . . . . .	32
4.11 Filtered table by PM's name (Trello) . . . . .	33
4.12 One of Trello Boards with assigned tasks . . . . .	33





# Chapter 1

## Introduction

Currently there are numerous more or less incompatible task tracking solutions – including task managing software - on the market. In teams where work is going parallel on different projects team members often use different task managers. On average, the developer may use up to 5 PM solutions, and to manage all his tasks he has to navigate (switch) between them fairly often.

The main idea is to create an app (ATT) that will combine the most common PMs and allows basic usage of them. The application should not provide full usability of the original PM, rather quick access to their most popular features. The original PM would provide better UI, amount of features e.t.c, but we suppose that in most cases developer will not use all of them - only basic ones. And this is why ATT will allow to display only assigned tasks and if a developer needs to find something else he would better use a specific PM where the task belongs. Also, ATT will allow some simple CRUD operations such as leaving a comment under a specific task, filtering tasks by name or by PM name. In the next chapters, I will describe the process of development in detail from research to user testing of the implemented application.



## Chapter 2

### Done research

In this chapter, I will describe the first steps which should be done before starting to design the system architecture or implementation phase. At this moment I expect to find out how to efficiently parse tasks from PMs, which technologies to use, and the approximate architecture of the application.

### 2.1 REST API

The first task to solve was to check if popular PMs provide REST API which we can use to fetch the task to the ATT application. In the sections below I will describe some of the most common PMs (Trello, Jira, Asana) and provide short description of them.

#### 2.1.1 Trello

Trello provides REST API which allows to set up webhooks and fetch requests and take data directly from Trello, so there is no need for parsing web pages, etc [2]. Because an API is provided by Trello developers we can suppose that it will be available in the long term future. Trello uses a delegated authentication and authorization flow so the application never has to deal with storing or handling usernames or passwords. Instead, it passes control to Trello (identifying itself via the API key) and once Trello has allowed the user to choose an account and sign in, Trello will hand the user and control back to the ATT application, along with an API token. To get started, the ATT application needs an API key. It can be obtained by logging into Trello and visiting <https://trello.com/app-key>. ATT users will always see the Trello authorization screen when granting ATT application access. The permissions, duration of access, and application name displayed are all configured via the URL parameters. Once a user clicks "Allow" he will grant ATT app access to his account and be redirected to a page that contains the API token. This token, along with his API key, can be used to read and write for the user's entire Trello account. Tokens should be kept secret.

### ■ 2.1.2 Jira

The Jira platform provides Java APIs that the ATT app can use to interact with Jira programmatically[9]. These APIs are common to all Jira applications. In addition, Jira Software and Jira Service Desk provide APIs for application-specific functionality. The Jira Java APIs are typically used when building Plugins2 apps (for Jira Server). The Jira REST APIs are used to interact with the Jira Server applications remotely, for example, when configuring webhooks. The Jira Server platform provides the REST API for common features, like issues and workflows. The Jira Software and Jira Service Desk applications have REST APIs for their application-specific features, like sprints (Jira Software) or customer requests (Jira Service Desk):

- Jira Software Server REST API
- Jira Service Desk Server REST API

### ■ 2.1.3 Asana

The Asana API is a RESTful interface providing programmatic access to much of the data in the system[6]. It provides predictable URLs for accessing resources and uses built-in HTTP features to receive commands and return responses. This makes it easy to communicate with from a wide variety of environments, from command-line utilities to gadgets to the browser URL bar itself. The API accepts JSON or form-encoded content in requests and returns JSON content in all of its responses, including errors. Only the UTF-8 character encoding is supported for both requests and responses. Pagination is an important concept when working with queries for multiple objects. Requests with large result sets may timeout or be truncated; therefore, pagination is strongly encouraged to ensure both you and your users have the best experience when using the Asana API.

## ■ 2.2 Authorization in provided REST API from Trello, Jira, Asana

For interacting with the REST API the user should allow the ATT application to access his account. After that, the application will store the PAT token locally (in local storage) and in the database for future usage.

### ■ 2.2.1 Trello

Trello's API uses token-based authentication to grant third-party applications access to the Trello API. Once a Trello user has granted application access to their Trello account and data, the application is given a token that can be used to make requests to the Trello API on behalf of the user. There are two ways to authorize a client and receive a user token. The first is via Trello's

1/authorize route, the second is via basic OAuth1.0. More information could be found on the Trello developers pages[4].

For the implementation of authorization in Trello from the ATT application OAuth 1.0 was chosen because it allows multi-users usage in the future. Trello provides a `client.js` library to make it easier to interact with Trello's API in third-party applications. It provides a global Trello object that includes helper methods for common actions in Trello's API. There are two steps to use the `client.js` library. The first is to get the ATT application key. This key identifies ATT to the API and is needed for any authenticated or unauthenticated API calls. Using the Trello object, we need to authenticate the user. This is done with the `Trello.authenticate` method. This method will automatically trigger Trello's authorization flow and return an authentication token. This token will be specific to the user's API key. This token grants access to the authenticated user's boards, lists, cards, and other settings, depending on the permissions requested in the authenticate method. We now can access our user's Trello data. The functionality provided via `client.js` makes use of callbacks for success and failure. The authorization flow is shown in more detail in the following diagram.

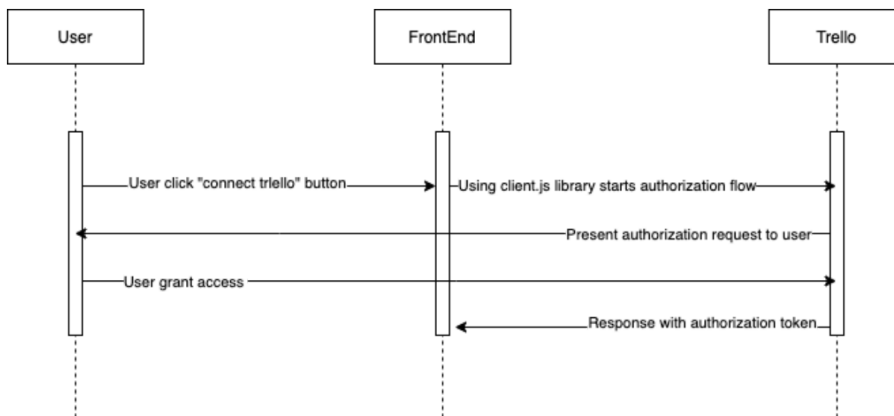


Figure 2.1: Trello authorization flow

### 2.2.2 Jira

Jira uses 3-legged OAuth (3LO), which means that the user is involved by authorizing access to their data on the resource (as opposed to 2-legged OAuth, where the user is not involved). More detailed information could be found on official Jira REST API documentation [10]. In Jira, a client is authenticated as the user involved in the OAuth dance and is authorized to have read and write access. The data that can be retrieved and changed by the client is controlled by the user's permissions in Jira. The authorization process works by getting the resource owner to grant access to their information on the resource by authorizing a request token. This request token is used by the consumer to obtain an access token from the resource. Once the client has an access token, it can use the token to make authenticated requests to the

resource until the token expires or is revoked. This process can be separated into three stages:

- 1. The user authorizes the client with Jira to receive an access code.
- 2. The client makes a request to Jira with the access code and receives an access token.
- 3. The client can now receive data from Jira when it makes a request including the access token. Note, the client can continue making authenticated requests to Jira until the token expires or is revoked.

The authorization flow is shown in more detail in the figure 2.2

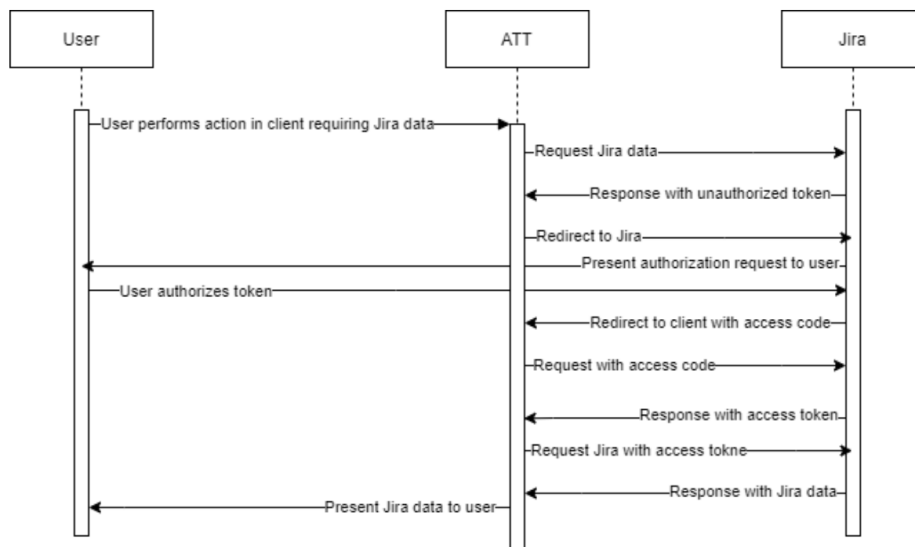


Figure 2.2: Jira authorization flow

### ■ 2.2.3 Asana

Asana supports a few methods of authenticating with the API. Simple cases are usually handled with a Personal Access Token, while multi-user apps utilize OAuth.

- OAuth 2.0 Asana requires that applications designed to access the Asana API on behalf of multiple users implement OAuth 2.0.
- Personal Access Token Personal Access Tokens are designed for accessing the API from the command line or from personal applications.

For ATT implementation was chosen authorization via OAuth 2.0 because the app can be used in the future for multi-users usage. The authorization flow is shown in more detail in the following diagram.

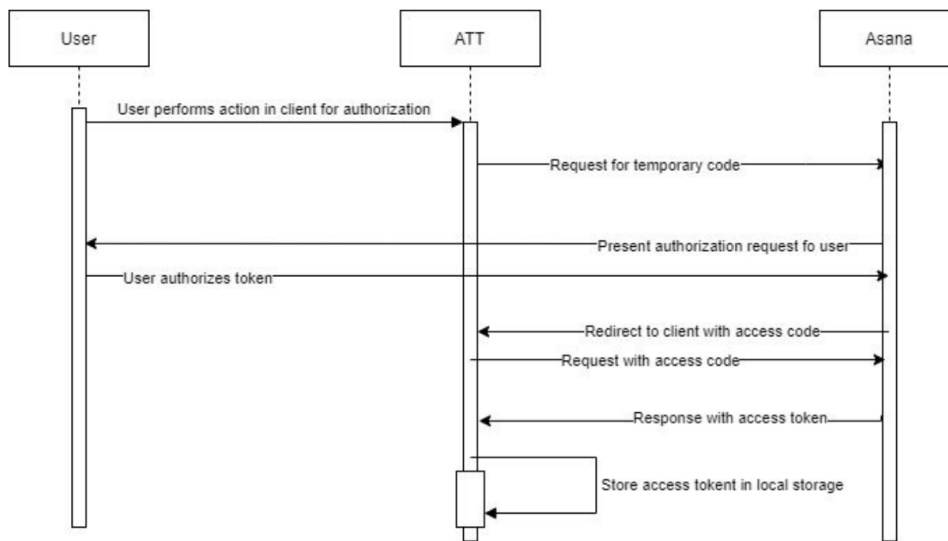


Figure 2.3: Asana authorization flow

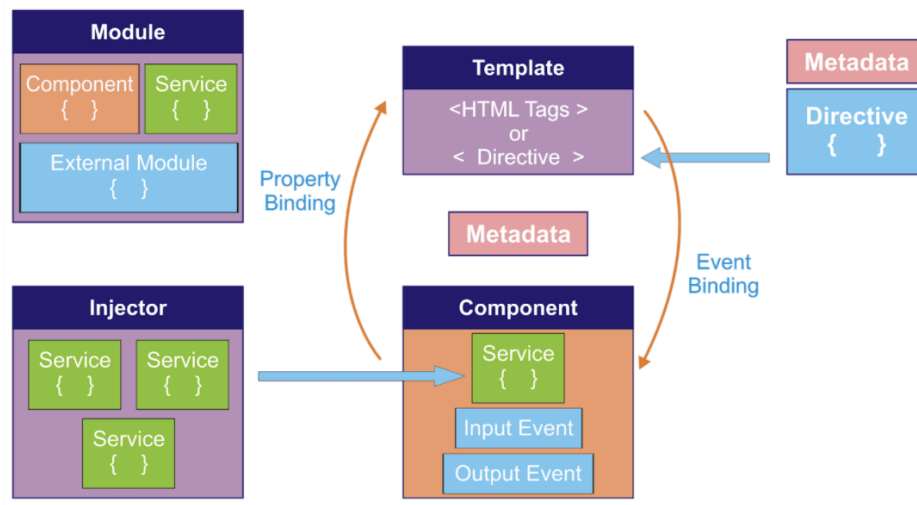
## 2.3 Fetching tasks from PM

As the ATT app is already authorized in all PM, authorization tokens for Trello, Jira, Asana are stored in local DB and LS. Now we can make API calls using them to get all cards assigned to the authorized user. More info about this in the next chapter (Implementation of fetching task section)

## 2.4 Used technologies

- Front end: Angular, HTML, SCSS, REST
- Back end: Java Spring
- Database: Postgres

First and the foremost, the ATT app should be a client-based app that will consist of a front-end application and connect to a database. For the front end, I wanted to choose one of the currently popular frameworks, Angular[1] or React JS[11]. As a style framework, I chosed Bootstrap[12]. At the end of the front end side, I decided to use Angular because it has better architecture - all parts of code are written separately (HTML, SCSS, TS). The figure 2.4 will describe the basic Angular Model and how its components communicate with each other.



**Figure 2.4:** Angular Model

On the part of connecting the database to the ATT app, I figured out that the database cannot be connected to the front end, because it would be insecure by creating an opportunity for intruders to manipulate the database and steal the private data of users and manipulate their accounts in the future. So the ATT app will need a back-end server application. As a back-end platform Java Spring Boot was chosen[13]. I stopped my view on it because of the features and benefits it offers as given here:

- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container

## 2.5 Summary

All task managers provide tools for other developers to make their products more popular and usable. In most cases, task managers insist on using OAuth authorization, differ only by the level of provided API. Trello provides the developers with client.js library which allows authenticating users in one command, the result will be Personal Access Token stored in local storage under `trello_token` variable. With Jira and Asana, all looks a bit different. With them, I should implement OAuth authorization in my way, because they



don't provide done solution for that. And also unlike Trello, I should use my backend part for authorization. With authorization flow for different task managers, you can get acquainted bellow.



# Chapter 3

## System architecture

In this chapter, I will describe system architecture, database model, and the base operations via sequential diagrams.

### 3.1 System overall diagram

On the figure 3.1 we can see the system overall diagram, which describes parts of the application and how they communicate. The base idea is - user interacts with ATT UI, the front-end part, which allows the to user create a new account, log in into the existing one, connect PMs, track the tasks, and perform some CRUD operations on tasks. It will communicate with external API (Trello, Jira, Asana) and with backend part via HTTP requests. The ATT app can perform actions with DB only via backend, for security reasons.

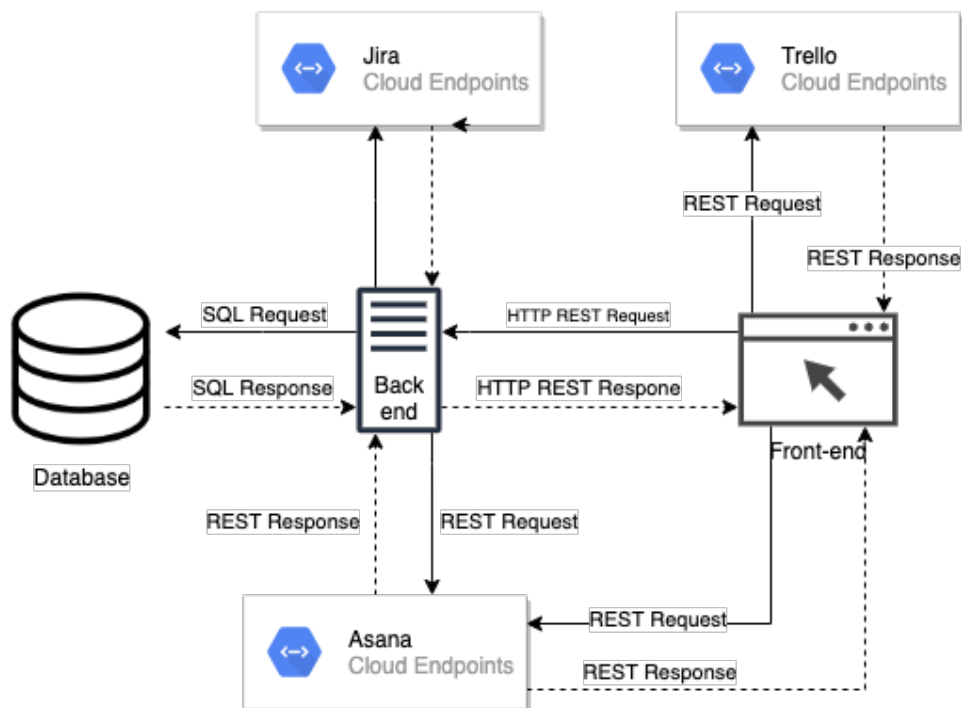


Figure 3.1: System overall diagram

## 3.2 Database UML diagram

Figure 3.2 describes the content of the database, in this case, there are only two tables "User" and "Token". The database was created for storing PAT for specific users and for storing users' data. Firstly user creates his account and his username, email, name, surname, the password will be stored in DB. After ATT app will connect PAT and the user, for which token was saved, for future manipulations with them such as updating PAT if it expired, adding new PAT e.t.c As we can see in this version of the DB table User has:

- Unique id
- Email, which can be used as login for successful authorization
- Name
- Password, which also should be used for successful authorization
- Surname
- Username, by default user has two options, use username or email for authorization

Table "Token" has:

- Unique id, which is used to connect specific token to specific user
- Name, which will allow to detect for which PM belongs
- Refresh token, which is used for Asana case, All other PMs provide PAT which won't expire within time. In case of Asana we should use refresh token to get new PAT every hour.
- Token, which is actually PAT

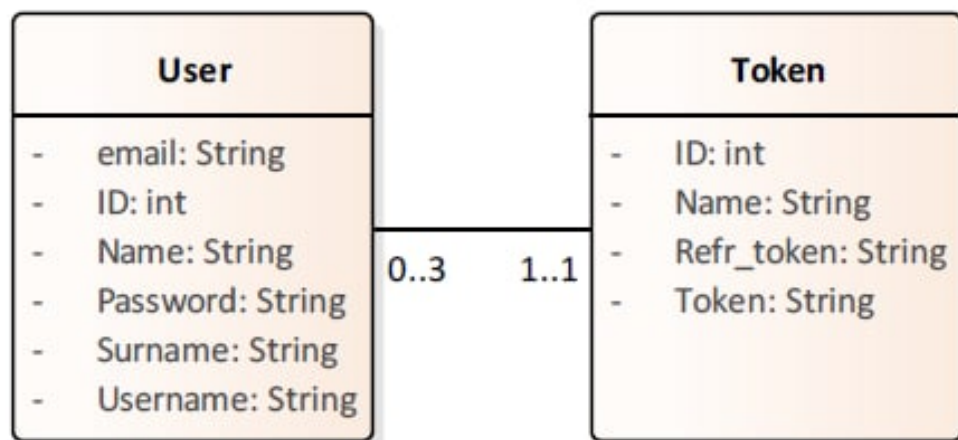


Figure 3.2: UML class diagram

### 3.3 Trello Authorization flow

This sequence diagram describes the Trello authorization flow [4]. Firstly user should start the action by clicking on **Connect** button, after a huge part of the work will be done by Trello `client.js` library. The user will only need to check if the displayed data is correct and grant access to his account from my ATT app. After that, the PAT will be stored in the local storage of the web browser and will be sent via HTTP POST request at the back end to be stored in DB.

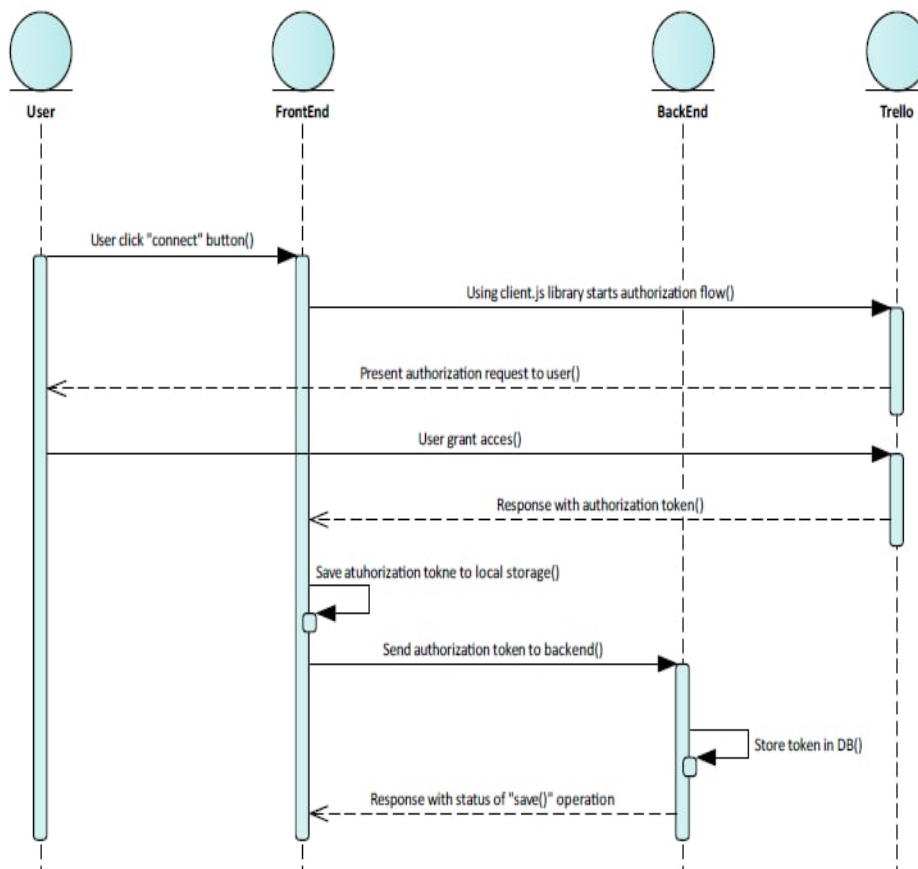


Figure 3.3: Trello Authorization flow

### 3.4 Jira Authorization flow

The idea is the same as with Trello, but the implementation is different. Firstly user starts authorization flow with click on **Connect** button. Then HTTP GET request will be sent to backend side for access URL, the user will be automatically redirected to granting access page, where should confirm that user provides access to his account to third party application (ATT app). He will be given a temporary code that should be pasted to a specific form

on the front-end side. After that, code will be added to an HTTP request to the back end, the back end will exchange it on PAT. PAT will be stored in DB and added as a response for previous HTTP request. Token will be saved in local storage in browser and on that point authorization flow is finished.

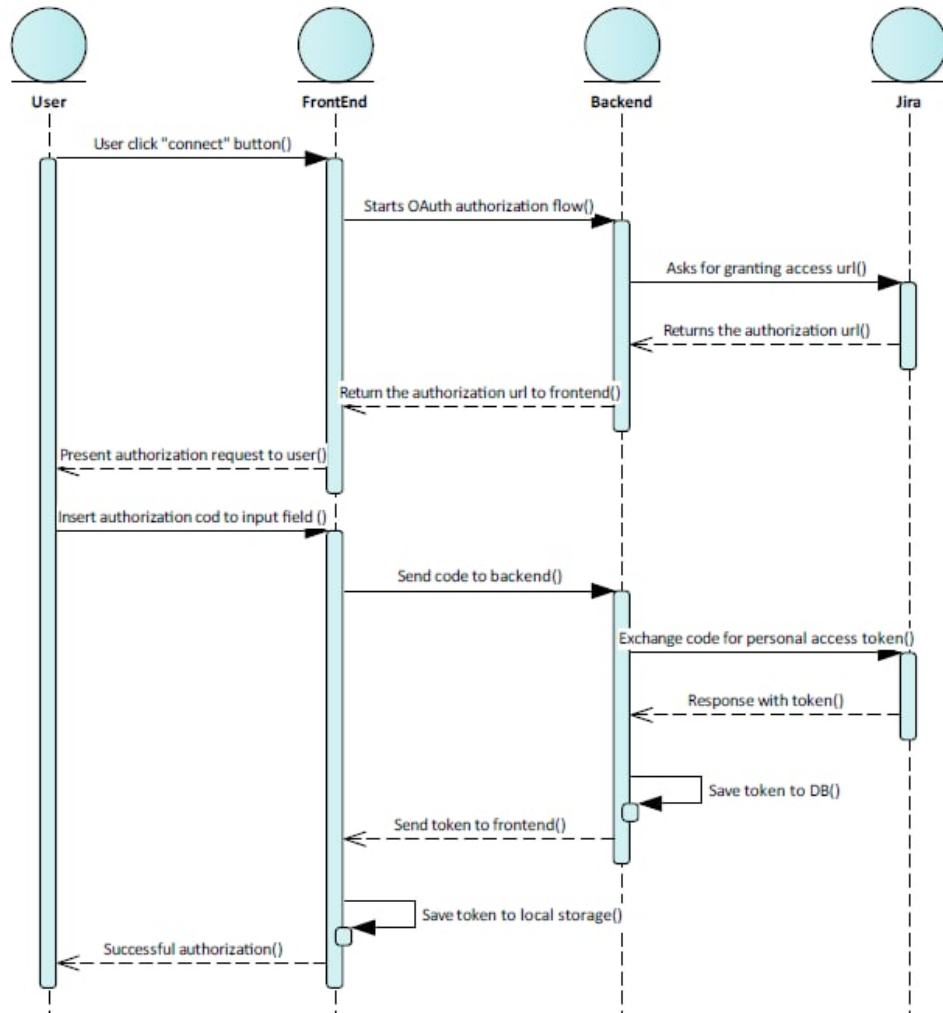


Figure 3.4: Jira Authorization flow

### 3.5 Asana Authorization flow

Authorization flow with Asana is the same as with Jira. User should start it with **Connect** button. After that, he will get access URL from ATT back end side and will be redirected to that page immediately. When the user confirms access, he will be redirected back to the user page, temporary code will be given as a query parameter in the URL. Then the front end will parse the URL, extract the code and create an HTTP GET request with it. When the back end exchange it to PAT, the token will be stored in DB and added as value to the HTTP response. After that, the token will be also stored in the browser's local storage.

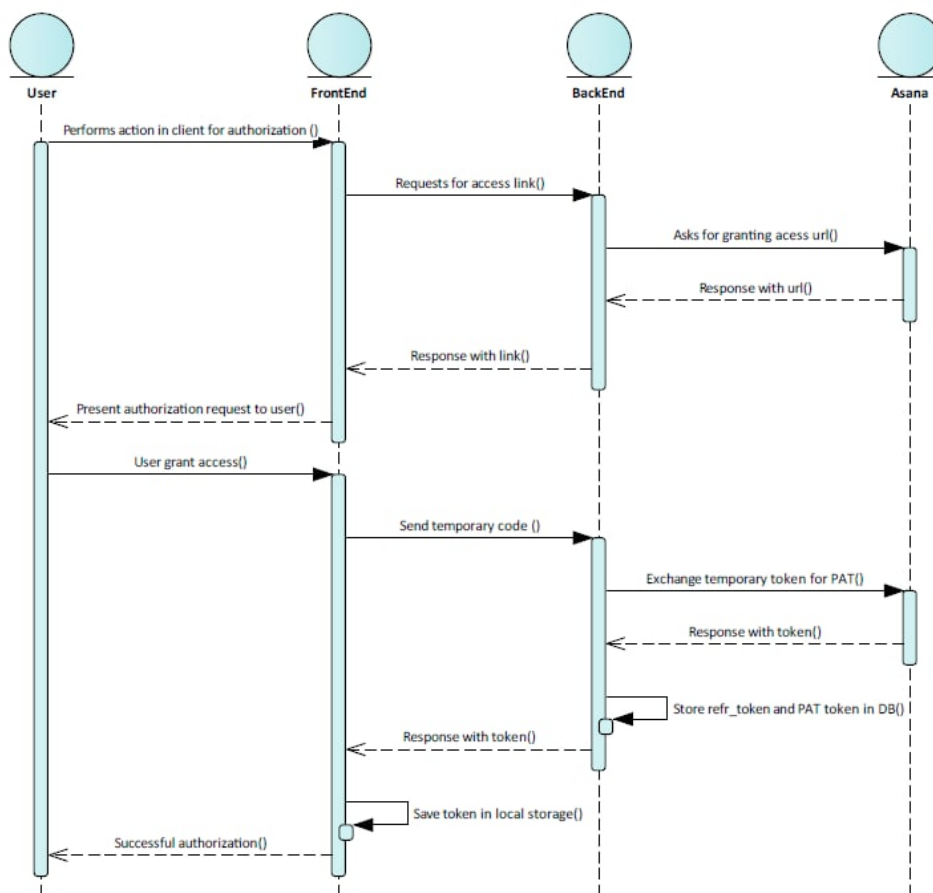


Figure 3.5: Asana Authorization flow





## Chapter 4

### Implementation

As a practical representation, I decided to build an application, that will allow testing all use cases and functionality. The prototype will support the following features: create a new account, log in to it, connect PMs, parse assigned tasks from PM, filter tasks by keyword, leave a comment which will be transferred to the original PM task/issue. It will be described in more detail in the following chapters dedicated to the implementation of particular parts of the application.

#### 4.1 Authorization

One of the most important parts of an application is security, so in this chapter, I will describe it in terms of different parts of the ATT app. The authorization process is divided into two parts - internal and external. Internal authorization is authorization in the ATT app using a username and password which the user defined during the registration process. The method of internal authentication can be changed, for example on a production server we can use the MyID library which will allow us to authenticate a user via his PKI card. External authorization is authorization and granting access to different task managers. The authorization flow of external authorization is shown above as diagrams, but here we will have closer look at it and I will describe why the final implementation looks different as it was described in the previous chapter.

#### 4.2 Internal Authorization

In this part, we will have a closer look at internal authorization and how it works on different levels of application. It is divided into 3 parts - Database, Back end, and Front-end. DB is responsible for storing and searching the data. It can be grant access, get contact info about the user, get all tokens that are connected to the user. The back end is responsible for security and DB manipulations and the Front end for the user interface.

### 4.2.1 Database

The role of the database in the authorization process is quite simple - store the username and hashed password for future usage.

### Registration

For creating a new user account, the user should fill the registration form in the front end (more in the front-end authorization section), and data from this form will be added to the database after validation.

### Sign in

After user creates his new account, he can login via his credentials, back end will check if user with this credentials really exists using the following SQL query for searching

**Listing 4.1:** SQL query for sign in method

```
@NamedQueries({
    @NamedQuery(name = "User.findByUsername",
        query = "SELECT u FROM User u WHERE u.username = :username")
})
```

Next it will return the user object and the back end will do the next step of authorization.

### 4.2.2 Front-end

The front end's role in authorization is: validate input data, build a JSON object, send it to the back end, save the JWT token eventually handle the exception.

### Registration

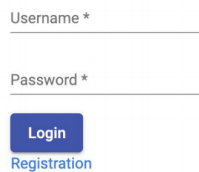
The image shows a registration form with four input fields, each followed by an asterisk to indicate it is mandatory. The fields are: Username, Email, Password, and Retype password. Below the fields is a blue button with the text 'Create'.

**Figure 4.1:** Registration form from frontend

For registration, the user should fill all mandatory fields (are marked with \*) Front-end will validate all inputs before sending data to the back end. First of

all, it will check if all inputs were user data, not a hostile script. It will prevent XSS, CSRF attacks. It will check if the email input field is a “@” sign and it is an email. And it will check if passwords are the same, have length > 8 symbols, etc. If verification was successful, data will be converted to JSON format and sent to the back end endpoint `http://localhost:8080/att/register` via HTTP POST method.

## ■ Sign in



Username \*

Password \*

Login

Registration

**Figure 4.2:** Login form from frontend

For sign-in user should fill all mandatory fields (all mandatory fields are marked with \* ) As for the registration part, the front-end will validate all fields before sending data to the back end and preventing XSS, CSRF attacks. After control data will be converted to JSON format and sent to `http://localhost:8080/att/authentication` endpoint. As a response, a JWT token will be received. This token will be used for future operations such as getting current signed user, storing user’s authentications tokens for PM. Also, this JWT token will be stored in local storage.

## ■ 4.2.3 Back end

The internal authorization back end is responsible for storing data in the database and for checking if a user with this username and password exists. For this version of the application, I used JWT based authorization. What does it mean? If a user with this combination of username and password exists in the database or after the successful registration user will get the JWT token, front-end will store it for future usage in local storage. After the user gets his JWT token the front end will automatically sign all his HTTP requests with it. The back end will get the HTTP GET/POST request with an authorization header and will validate the JWT token from there. If the token is valid - HTTP request will be processed, if not - the user will get a 403 error - “Forbidden error”

## Registration

**Listing 4.2:** Endpoint for creating new user

```

@RequestMapping(value="/register", method = RequestMethod.POST)
public String register(@RequestBody StoreUserDto userDto) {
    if(userService.findByUsername(userDto.getLogin())==null) {
        userService.store(userDto);
        return "200";
    }
    return "401";
}

```

For the creation of a new user account, the back end should check if a user with the same username is not already in the database. It will call the `.findByUsername` method from `userService`, which is calling `.findByUsername` from `userDao` and is implemented like that.

**Listing 4.3:** Endpoint for creating new user

```

public User findByUsername(String username) {
    try {
        return em.createNamedQuery("User.findByUsername",
            User.class).setParameter("username", username)
            .getSingleResult();
    } catch (NoResultException e) {
        return null;
    }
}

```

After that, the user will be stored in the database and can login into his new account.

## Sign in

**Listing 4.4:** Endpoint for creating new user

```

@RequestMapping(value = "/authentication", method =
    RequestMethod.POST)
public ResponseEntity<?> login(@RequestBody AuthenticationRequest
    authenticationRequest) throws Exception {
    try {
        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                authenticationRequest.getUsername(),
                authenticationRequest.getPassword()
            ));
    } catch (BadCredentialsException e) {
        throw new Exception("Incorrect username or password", e);
    }
    final UserDetails userDetails = userService.
        loadUserByUsername(authenticationRequest.getUsername());
}

```

```

    final String jwt = jwtTokenUtil.generateToken(userDetails);
    final Integer id=userService.
        findByUsername(userDetails.getUsername()).getId();
    return ResponseEntity.ok(new AuthenticationResponse(jwt, id));
}
}

```

---

After the back end gets a JSON object with a username and his password, it should verify if the user with this username and password exists. For that, I use `.authenticate` method from `authenticationManager` and if a user exists will be sent the response with JWT which will be stored in `localStorage` in the browser and will be used for future requests to the back end.

## 4.3 Authorization in PM's

Authorization in PMs is a different process from internal authorization because we should provide all data that we need for successful and comfortable usage of PM's API. First of all, the authorization flow is specific for different PMs. For example, to be able to use Trello API we should connect `client.js` library from Trello developers and call the method `Trello.Authenticate()`. But for Jira we can't use the front-end, so we should perform OAuth dance on the server part. The process is the same for Asana. Authorization in different PMs will be described deeper in the next sections.

### 4.3.1 Database

Database role in external authorization (Authorization in PMs) is the same as in internal - store the PAT token for future usage. After the user grants access to his account on the front-end part, the token will be sent to the back end and saved. This is need for the prevention of data loss and successful re-authorization.

### 4.3.2 Front-end

One of the main front-end roles is to connect user actions and the back end. The behavior of the front-end for different task managers is different, but the goal was to create a comfortable UI that will require minimum user interaction. I will describe the differences in the next few sections.

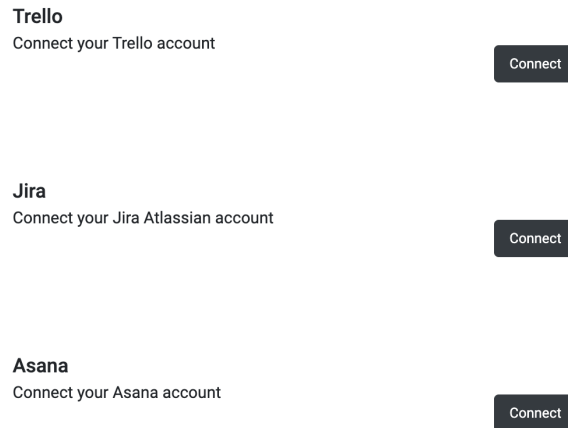


Figure 4.3: UI for account PM connections page

## Trello

When the user clicks on **Connect** button the external authorization in Trello will begin. As I described above, Trello developers provide us a `client.js` library for this purpose. So I added a specific method `Trello.authorize()` which is called after the user will click on the button. After that user will be redirected to the access granting page.

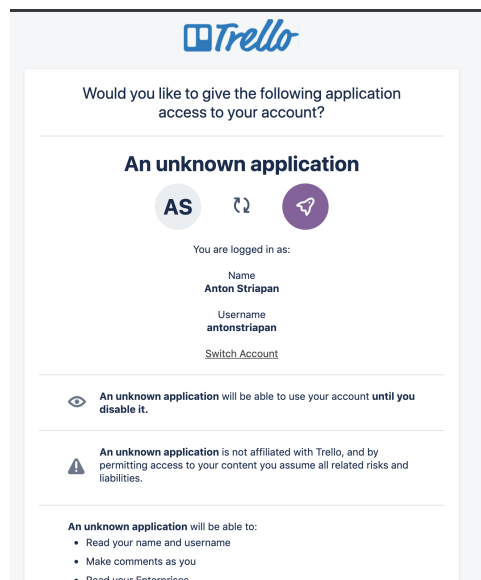


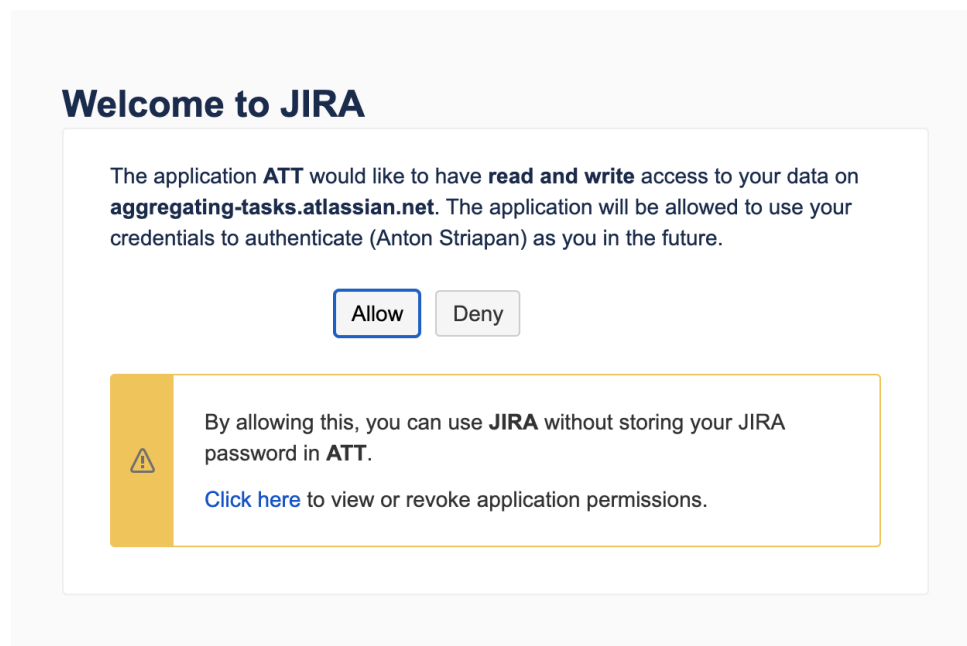
Figure 4.4: Trello granting access page

When the user confirms that he is giving access to his account to the third level application he will be redirected back and the PAT token will be saved

in local storage. After the front end detects the PAT in local storage it will send it to the back end via an HTTP POST request, where it will be saved for future usage. For example, when a user signs into his account, the front-end will send the HTTP GET request to get all PAT tokens assigned to the user.

## ■ Jira

In the case of Jira, we can't perform API calls from the front-end because of the CORS settings of Jira. This limitation doesn't apply to the full "paid" version of Jira, but in our case - we had to find the solution for this shortcoming. It was solved by implementing authorization flow on the back-end side and use the front end only as a tool which is connecting the user to the back-end. As in Trello, the user should start authorization flow with **Connect** button. After that front end will send HTTP GET request to `/tokens/jiraAuth` endpoint and as a response it will get the granting access URL. Then the user will be automatically redirected to that page where he should confirm providing access to the ATT app.



**Figure 4.5:** Jira access granting page

Then user should copy the code and paste it into form in the ATT app.

## Jira

Connect your Jira Atlassian account

Paste token here \*

---

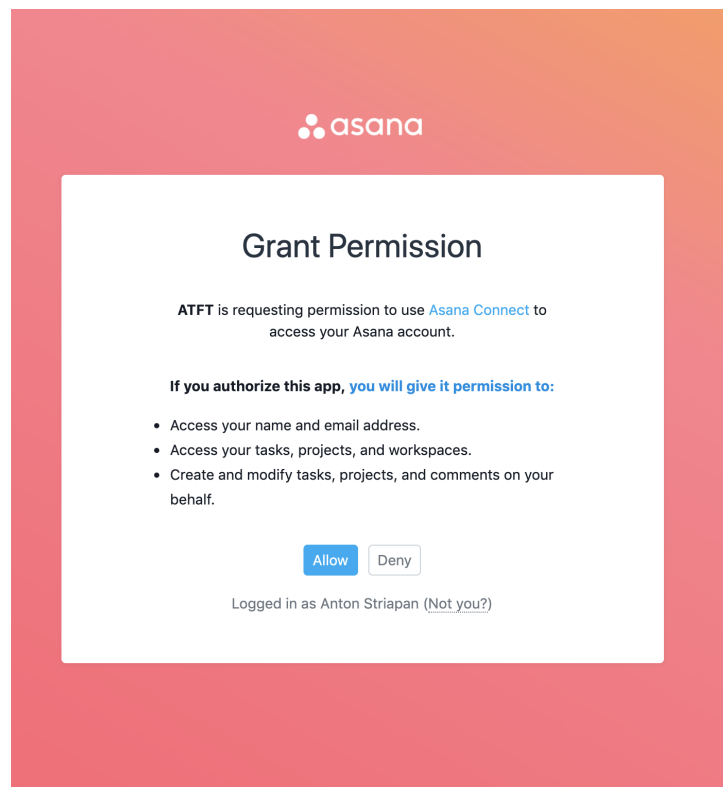
  

**Figure 4.6:** Jira form for code

After that, the front-end will send an HTTP POST request to the back end and as response, it will get the PAT token, which will be stored in the local storage.

## Asana

Asana had a ready-made solution for implementing authorization in Java, so the authorization was moved to the backend side. Because of that the user should only click on **Connect** button and the front-end will do the rest - it will send HTTP GET request to the back end for granting access URL.



**Figure 4.7:** Asana access granting page

When the response will be parsed, the user will be redirected to the access



granting page where he should confirm the action. After that, the user will be redirected back with the access code given as a query parameter in the URL. The internal parser will detect that the user was returned and the access code is present. The code will be extracted and sent as a body parameter to the back end. As a response client will get the PAT for Asana, already stored in DB, and store it in local storage on the front-end.

### 4.3.3 Back end

In external authorization back end is used as a proxy because most PMs (Asana and Jira) don't allow CORS requests. The main idea is to store the PAT token in the database, but before this, we need to start OAuth dance to get a temporary code, then exchange it for an access token. In the case of Asana, we also will get a refresh token. For different PM's we need various approaches in the implementation of authorization. I will describe all problems and solutions in according subsections below.

#### Trello

Back end doesn't play any role in authorization in Trello, because Trello developers provide us a front-end library for it, so all authorization flow is done on front-end side and we use back end only for storing PAT by sending the POST request on `http://localhost:8080/att/tokens/user-id`

#### Jira

Firstly user should start his OAuth dance by triggering `/jiraAuth` endpoint. It will return an access link, where the user should confirm that he is granting access for his account to the ATT application.

**Listing 4.5:** Endpoint for getting granting access link

---

```
@GetMapping("/jiraAuth")
public ResponseEntity<?> jiraAuth() throws Exception {
    return ResponseEntity.ok(new
        AuthResponse(tokenService.jiraAuth()));
}
```

---

After that, he will get the code which he should paste to the front-end form, submitting this form will trigger `/jiraAuth-id` endpoint. As id it has the id of a user who has started the authorization flow and in the body of the HTTP request it has code, which was pasted by the user.



```

    }else{
        code=code.substring(0,code.length()-7);
        accessToken=app.fetchToken(code);
        refreshToken=app.credential.getRefreshToken();
    }
    String refreshToken=app.credential.getRefreshToken();
    token.setToken(accessToken);
    token.setRefreshToken(refreshToken);
    if(tokenPersists(id,"Asana")==false){
        persist(token);
    }else{
        List<Token> tokenList=tokenDao.findAllForUser(id);
        for (Token key:tokenList){
            if(key.getName().equals("Asana")){
                key.setToken(accessToken);
                tokenDao.update(key);
            }
        }
    }
    return accessToken;
}

```

After this step was done, we will have PAT and refresh token stored in our DB and also the PAT will be send to the front-end side.

## 4.4 Fetching tasks from PM

When the authorization step is done, we can move forward and try to fetch some data from PMs using their REST API. The main idea is to parse only task assigned to the signed-in user - there is no need to fetch all possible tasks. The application should provide as much data as it is possible, so the application will fetch - description, members of the task, comments, because in comments may be stored critical information, and of course, due date. I will try to explain all the details in the next sections.

### 4.4.1 Trello

When the Trello account is connected to the ATT app, there were no issues with getting all cards which were assigned to authenticated users.

**Listing 4.9:** Fetching tasks from Trello

```

if (localStorage.getItem('trello_token') != null) {
    this.http.get(
        "https://api.trello.com/1/members/me/cards?key="+this.appKey
        +"&token=" + localStorage.getItem("trello_token"))
        .subscribe(
            data => {
                for (let i in data) {
                    let temp = {

```

```

        projectManager: "Trello",
        name: data[i].name,
        due: data[i].due,
        id: data[i].id,
        desc: data[i].desc
    }
    this.listOfAllTasks.push(temp);
}
}
);
}

```

First of all, the ATT application should check if the Trello PAT is in local storage or not. After that, using the endpoint from the example above, the ATT app will get all cards assigned to the user. As query parameters it has `key` - this is the API key of the ATT application (which allows making API calls) and `token` - is Personal Authorization Token (PAT) which shows that the user is authorized and allows access to his data in Trello. After the API call is done ATT app will serialize the Trello issue object and add it to `listOfAllTasks`.

#### 4.4.2 Jira

When the Jira account is connected to the ATT app, There were a lot of issues with getting all cards assigned to authenticated Jira account users.

- Because the authentication part is based on the back end side, the front-end (browser) didn't store the cookies indicating that the user was authorized. It makes API calls from the front-end side impossible.
- There were new rules for authentication requests. I spent a lot of time figuring out how I should add a personal token to the requests, so they could pass.
- Problems with building AQL query to get only cards which are assigned to me
- Problem with CORS

The final solution was to build a personal endpoint on the ATT back end, allowing to make authenticated API calls.

**Listing 4.10:** Fetching tasks from Jira

```

if (localStorage.getItem("jira_token") != null) {
    this.http.get<any>("http://localhost:8080/att/jira/tasks", {
        headers: {
            Authorization: `Bearer ${this.authService.getToken()}`,
            Accept: "application/json"
        }
    }).subscribe(
        data => {

```

```

    for (let i in data.issues) {
        let temp = {
            projectManager: "Jira",
            name: data.issues[i].fields.summary,
            due: data.issues[i].fields.duedate,
            id: data.issues[i].id,
            key: data.issues[i].key,
            desc: data.issues[i].fields.description,
            members: [data.issues[i].fields.assignee.
                    displayName]
        }
        this.listOfAllTasks.push(temp)
    }
}
);
}

```

It is a regular request to ATT back end side, with no parameters, except of the JWT token in the request header. Jira Controller calls `JiraService.getAllAssignedTasks()` method which is implemented as shown on listing above.

**Listing 4.11:** Fetching tasks from Jira

```

@Transactional
public String getAllAssignedTasks() throws Exception {
    JiraAuth jiraAuth=new JiraAuth();
    JSONObject json = new JSONObject(jiraAuth.auth(new String[]{
        "request",
        "https://aggregating-tasks.atlassian.net/rest/gadget/1.0/currentUser"
    }));
    String query="assignee="+json.getString("username");
    query= URLEncoder.encode(query,"UTF-8");
    String tasks = jiraAuth.auth(new String[]{
        "request",
        "https://aggregating-tasks.atlassian.net/rest/api/2/search?jql="+query});
    return tasks;
}

```

First, it makes the first API call to get a current authorized user, then builds a new query with the username and makes a new API call to get assigned tasks. When the front end gets a response with all tasks, it is added to the list with Jira cards, which will be displayed later.

### 4.4.3 Asana

When the Asana account is connected to the ATT app, there were no issues with getting all cards that were assigned to authenticated users. The first step was to get all workspaces the user has access to, then using their id's make a new API call to get cards from them and add the cards to the list of Asana cards. The difficulty was to synchronize those API calls because usual requests are made asynchronously. I implemented one API call inside

another and when the last one will be made the data will be serialized and added to the list.

**Listing 4.12:** Fetching tasks from Asana(back end side)

---

```




if (localStorage.getItem('asana_token') != null) {
  this.http.get<any>("https://app.asana.com/api/1.0/workspaces", {
    headers: {
      Authorization: 'Bearer
        ${localStorage.getItem("asana_token")}',
    },
    observe: 'response'
  }).subscribe((response) => {
    let data = response.body.data
    for (let i in data) {
      this.http.get<any>(
        "https://app.asana.com/api/1.0/tasks?assignee=me&workspace="
        + data[i].gid, {
        headers: {
          Authorization: 'Bearer
            ${localStorage.getItem("asana_token")}'
        }
      }).subscribe(
        data => {
          for (let card of data.data) {
            let temp = {
              projectManager: "Asana",
              name: card.name,
              id: card.gid
            }
            this.listOfAllTasks.push(temp)
          }
        }
      )
    }
  },
  (error) => {
    if (error.status !== 200) {
      this.tokenHasExpired = true;
      localStorage.setItem(
        "tokenHasExpired", this.tokenHasExpired.toString());
    }
  });
}

```

---

## 4.5 Displaying Tasks in front-end

The most important design goal of the front-end UI was the ability to display tasks from all task managers on a single page, without making the impression of being overloaded or confusing. The first version of the design displayed tasks as clickable cards showing full task information upon mouse click. This idea was later abandoned as cards didn't provide optimal filtering and searching functionality. The last version of the front-end displays tasks as an accordion control element comprising a vertically stacked list of tasks. Each item can be expanded or collapsed to reveal the details of the given task. This control provides also a filtering option.

NAME	DUE	PROJECTMANAGER
Create a new list from template, based on VoIP list		Trello
Create new tab, same as Multiuser (just copy it)		Trello
Implement actions as defined in document		Trello
Check E-mail address for Circuit telefonie against SCD		Trello
Database design		Trello
Server		Trello
Continue adding elements from the design document (Styleguide)		Trello
Tile component library (WCS Libraries)		Trello
 Finish setting up your first project		Asana
 Invite your teammates and start collaborating		Asana
 Download Asana's mobile app to manage projects on the go		Asana
NEW TEST JIRA		Jira
Test Assigned issue		Jira

**Figure 4.8:** Filled table with tasks from Trello/Jira/Asana

After some initial experiments, I have decided to optimize the task fetching process described in the previous sections. To reduce the time getting the task overview, properties of task (list of assigned users, comments, and description) visible only after opening the detailed look, are fetched only when needed – i.e. after clicking the affected section of the task list accordion control by the user.

## 4. Implementation

Filter		
Trello		
NAME	DUE	PROJECTMANAGER
Create backend (Flask)		Trello
<b>Description</b>		
Using Python Flask create backend which will allow communication with Jira API		
<b>Members</b>		
• Anton Striapan		
<b>Comments</b>		
Write a comment		
<input type="button" value="Save"/>		
Make deploy sereach		Trello
Front End implementation		Trello
Set up jira connection		Trello
Database design		Trello
Create a new list from template, based on VoIP list		Trello
Create new tab, same as Multiuser (just copy it)		Trello
Implement actions as defined in document		Trello

Figure 4.9: Opened task with description

Filter		
NAME	DUE	PROJECTMANAGER
Create a new list from template, based on VoIP list		Trello
Create new tab, same as Multiuser (just copy it)		Trello
Implement actions as defined in document		Trello
Check E-mail address for Circuit telefonie against SCD		Trello
Database design		Trello
<b>Description</b>		
Create the database design for youtube-automation application		
<b>Members</b>		
• Anton Striapan		
<b>Comments</b>		
Write a comment		
<input type="button" value="Save"/>		
• Ian Khartsev : - Local ID - Jira ticket ID - YouTube ID - Status: Approved/Not Approved - Approval comment		
Server		Trello
Continue adding elements from the design document (Styleguide)		Trello
Tile component library (WCS Libraries)		Trello

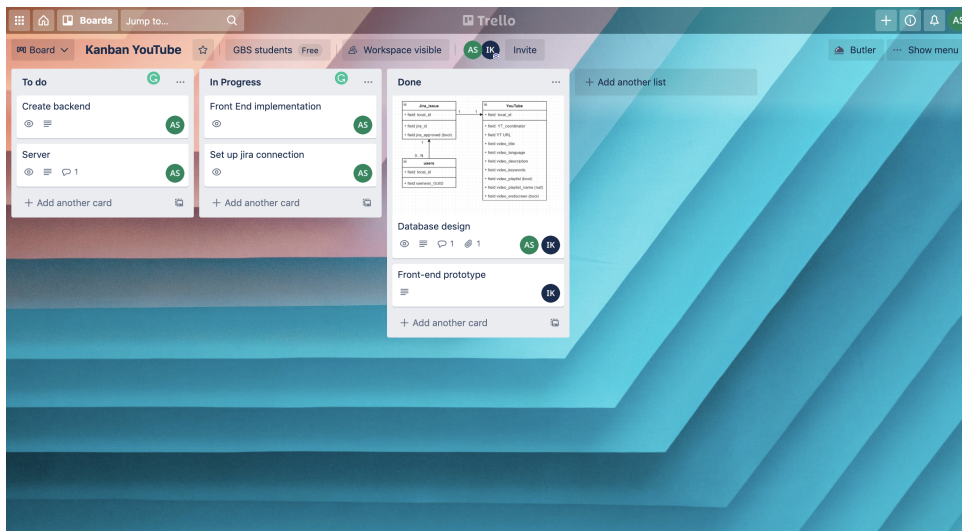
Figure 4.10: Opened task with filled comments



NAME	DUE	PROJECTMANAGER
Create backend (Flask)		Trello
Make deploy sereach		Trello
Front End implementation		Trello
Set up jira connection		Trello
Database design		Trello
Create a new list from template, based on VoIP list		Trello
Create new tab, same as Multiuser (just copy it)		Trello
Implement actions as defined in document		Trello
Check E-mail address for Circuit telefonie against SCD		Trello
Continue adding elements from the design document (Styleguide)		Trello
Tile component library (WCS Libraries)		Trello

**Figure 4.11:** Filtered table by PM's name (Trello)

As we can see in the figure above - there is a table displaying assigned tasks by PM's name, in our case - Trello. The figure below shows us one of the Trello boards with tasks. Assigned tasks to me are marked by green circle and initials AS in it.



**Figure 4.12:** One of Trello Boards with assigned tasks





## Chapter 5

### Testing



#### 5.1 Artem Hurbych (CTU, FEE, OI 3rd year student) first version testing

During the UX testing performed by the author, several issues were identified. Firstly, author pointed out on validation issues in login and registration pages. The application didn't have any validation for input with spaces and etc. However, as the application was in a development phase, this was acceptable for the time being. Secondly, there were no error alerts. Also, there were certain problems with Asana tasks, user should wait before task will be displayed, and there were issues with displaying connection status to task managers. No other issues were not found. The application looked promising and potentially useful.





## Chapter 6

### Conclusion

In the following section, I would like to sum up all my experience and problems which I faced in this project. In the beginning, the ATT app was planned as a client only app implemented on the Angular framework. However, architecture which makes direct access to DB from the front-end may cause a lot of security problems, and because of that I created back end side of the ATT application. As back-end technology, I choose Java Spring, because I have a lot of experience with it. After the back end was done, I started implementing connections to different task managers in order to acquire and store their access tokens ( PAT ).

With Trello there weren't problems, documentation for their API was very detailed. Documentation for Jira was not so exhaustive, as for Trello, so I had many issues reaching the above goal. First of all, Jira is not a centralized platform like Trello, it is a product, which creates a new instance for every team or organization that uses it. According to this, I set it up just for one instance of Jira, but in the future, I would like to add an option to connect it to different Jira workspaces to expand the ATT use case. The second difficulty was the strange OAuth "dance". Firstly user starts on the front end, then the back end generates an access link and sends it to the front-end, user grant access to his personal data from the ATT app, and get the code, which user should paste in a specific form and send it to back end after back end verify the code, it stores and send to front-end PAT. So user should initiate the process on the front-end, then the front-end will send HTTP request on back end endpoint for Jira authorization and response with access link, the user will be redirected to granting access page where he will get a code, which he should copy and paste on a specific form in ATT. After that will be the second part of authorization, the front-end will change that code on PAT. Asana also provides a library, which we can use to simplify getting PAT. It is connected to it via dependency in `pom.xml` When all tokens were saved in DB and LS, we can make API calls using it to get issues/tasks assigned to users.

For now, my application provides the following functionality:

- New user registration
- Login
- Connecting all PM ( Trello, Jira, Asana )
- Fetching tasks from PM
- Displaying all assigned to user tasks
- Providing more details to users about the displayed tasks, user can click on the row with the tasks and it will expand and display more information, such as description, assigned members and comments
- User can leave the comment which is then sent to original PM via API
- Fixed a “token was expired” issue, which happens because Asana provides a token, which can be usable only for a limited time.

Finally we can declare, that the goal the goal of this bachelor thesis was achieved - I created an application that combines 3 PMs in one. UI is user friendly and understandable for every user, it is not overloaded with unnecessary and potentially distracting information and provide basic operation with tasks. Usability of the application will be extended in the future to allow to connect more PMs, more operations with it, or connect all tasks to the user’s Outlook calendar to increase the effectiveness.

## Appendix A

### Application setup instructions



To start using the application firstly you should clone the git repository from QR code above (also you can click on it). Instructions how to run and serve the application you can find below.

## ■ A.1 How to run and serve application

- Open `application.properties` file
- Set up you database there
  - You should have Postgres installed, when it is done open the Postgres command line
  - Add your credentials to 10/11 lines in `application.properties` file
  - Use `create database att;` command to create the DB which will be connected to the ATT application
- Run the `att.java` in the `back/src/cz/att` folder
- Try to open `http://localhost:8080/att/current` for initializing script which will create all tables (this step is need only in first set up)
- You should have Node.js and npm package manager
- You can download Node.js from (<https://nodejs.org/en/>), it will install Node.js and npm
- Then run the `npm install -q @angular/cli` command in terminal window
- Open the `front` folder with your favorite ide/text editor (my choice is VS code)
- Use `npm i` command for installing all dependencies from `package.json`
- To start the application use `ng serve -o`





## Appendix B

### List of Abbreviations

- CRUD - Create Read Update Delete
- API - Application Programming Interface
- REST - Representational State Transfer
- HTML - HyperText Markup Language
- SCSS - Sassy CSS
- CSS - Cascade Style Sheets
- SQL - Structured Query Language
- JSON - JavaScript Object Notation
- JWT - JSON Web Token
- XSS - Cross-Site Scripting
- CSRF - Cross-Site Request Forgery
- DB - Database
- PM - Project Manager
- CORS - Cross-Origin Resource Sharing
- UI - User Interface
- ATT - Aggregation Tasks from different Task managers application



## Appendix C

### Bibliography

- [1] Angular project website [online]. ©2010-2020 Google [27.12.2020]. <https://angular.io/>
- [2] Trello Developer Guides [online]. ©2020 Atlassian [15.12.2020]. <https://developer.atlassian.com/cloud/trello/rest/api-group-actions/>
- [3] API Introduction [online]. ©2020 Atlassian [15.12.2020]. <https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/s>
- [4] Authorizing With Trello's REST API [online]. ©2020 Atlassian [15.12.2020]. <https://developer.atlassian.com/cloud/trello/guides/rest-api/authorization/>
- [5] Jira Server Developer Guides [online]. ©2020 Atlassian [17.11.2020]. <https://developer.atlassian.com/server/jira/platform/rest-apis/>
- [6] Asana Developers - Projects [online]. ©2020 Asana, Inc. [23.11.2020] <https://developers.asana.com/docs/projects>
- [7] Jobin Kuruvilla. Jira Development Cookbook [online]. Packt Publishing Ltd, ©2016 [17.11.2020]. <https://books.google.sk/books?id=ioZcDgAAQBAJ>
- [8] Hibernate One to Many Annotation Tutorial [online] <https://www.baeldung.com/hibernate-one-to-many>
- [9] Jira Rest API description ©2021 Atlassian <https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/>
- [10] Authorization in Jira REST API with OAuth ©2021 Atlassian <https://developer.atlassian.com/server/jira/platform/oauth/>
- [11] ReactJS © 2021 Facebook Inc. <https://reactjs.org/>
- [12] Bootstrap documentation <https://getbootstrap.com/>

C. Bibliography

---

- [13] Java Spring Boot documentation © 2002 - 2021 Pivotal, Inc. All Rights Reserved. <https://docs.spring.io/spring-framework/docs/current/reference/html/>
- [14] Angular Material Powered by Google LLC ©2010-2021. <https://material.angular.io/>