|  |
| --- |
| Ana Markovska, Martin Nikolov, Nikola Masalkovski, Petar Partaloski  21.02.2023 |

|  |
| --- |
| StudentBox Technical Documentation |

A picture containing box, container

Description automatically generated

# General

This introduction provides an overview of the Technical Documentation for StudentBox – a Web Administration Application. It includes the purpose, design approach, environment, main component design and high-level system design considerations.

## Purpose of the document

The purpose of this document is to define all the technical details of the system that we will be building. We will go though the process of explaining all the parts of our system to give the reader of the document a representation of what the system that we are building will the goals to achieve are and which services will be provided by it.

## Definitions and Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
| API | Application Programming Interface |
| ER Diagram | Entity-Relation Diagram |
| ORM | Object Relational Mapper |
| UI | User Interface |
| UX | User Experience |
| JWT | JSON Web Token |

## Reading instructions

TODO - EXPLAIN THE MAIN SECTIONS OF THE TECHNICAL DOCUMENTATION

## 

# Architectural Overview

There are several architectural approaches that were taken in our solution, considering many factors that could affect the system to make the solution better in every aspect.

StudentBox – Web Administration Application will follow a N-Layer Architecture so that each layer only depends on the layer below it.

Diagram, schematic

Description automatically generatedOur development team will develop their own API and the data will be stored in a relational database, the Front-end part - which will be developed using React, will be developed by the same team, where tasks will be regularly split equally.

1.High Level design

## Controller

In our application, we will be using REST APIs to communicate between our Back-End and Front-End applications. So we will then have Controllers that will handle all of the requests to our API. In this section, controllers are exactly what we will be focusing on.

In the next subsection, we will be displaying all the response models that are planned to be used. We define response model as a structure of data, where we will expect certain mappings of our API to respond to the calls with the data being organized in a predetermined structure that will be universal for all variations of calls for each API method.

|  |  |  |
| --- | --- | --- |
| Response Models | | |
| Name | **Description** | **Structure** |
| AuthorizedUser | A response model used in the process of Authorization, containing an access token and a refresh token, the access token can be used to make request, while the refresh token is used only to request for a new access token. | {  accessToken: String,  refreshToken: String  } |
| User | A response model to provide basic information about a certain user. | {  name: String,  lastName: String,  email: String,  avatarUrl: String,  skills: Skill[],  jobPositions: JobPosition[],  employmentInfo: EmploymentInfo,  educationInfo: EducationInfo  } |
| Company | A response model to provide basic information about a company. | {  name: String,  email: String  } |
| Post | A response model that is to be used for the representation of posts, contains HTML text and a Title, as well as the likes and replies to the post. | {  id: String,  title: String,  content: String,  likes: Integer,  comments: Reply[],  author: User  } |
| PostInfo | A response model that is to be used as a preview for a post, contains HTML text and a Title. | {  id: String,  title: String,  content: String,  author: User  } |
| Reply | A child-response model used to represent a reply, that can be a reply to a post. | {  id: String,  content: String,  likes: Integer,  author: User  } |
| Skill | A model to describe a skill that a certain user can possses. | {  id: String,  name: String  } |
| JobPosition | A model to describe the user’s preferred job positions that they could be looking for. | {  id: String,  name: String  } |
| EmploymentInfo | A model to provide more information about a user’s employment. | {  id: String,  companyName: String,  jobPosition: String,  startedAt: DateTime,  yearsExperience: Integer  } |
| EducationInfo | A model that represents basic information about a user’s education. | {  id: String,  schoolName: String,  startedAt: Date,  endedAt?: Date,  description?: String  } |
| SkillCertificate | A model that is used to represent a certificate that a user has for a certain skill. | {  id: String,  issuedBy: String,  issuedAt: Date,  description?: String,  documentUrl?: String  } |
| Employment-Suggestion | A model to represent an employment suggestion that a company account has published. | {  id: String,  company: Company,  jobPosition: JobPosition,  salary?: Long,  description?: String  } |

And, in the next subsection, we’d like to focus on the controllers we mentioned above would be handling all of the API requests. The controllers that our application will have will be:

* **User Controller** - A controller that will manage the parts of our application that the students, the first type of users that would require the activities such as registering, logging in, adding details about themselves and etc.
* **Company (Enterprise) Controller** – A controller that will manage the parts of our application that the enterprise users of our application would care about. Activities would include, registering, logging in, posting job offers and etc.
* **Job Position Controller** – A controller that is to be used to view and create new Job Positions in our system.
* **Posts Controller** – Used for all blog related activities, such as reading, posting, commenting, liking blog posts.
* **EmployMe Controller** – Used for the goal of our most unique feature that is yet to be found anywhere on the real-world market, a section of our application that connects employers and employment-seekers in the tech sphere.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| User Controller | | | | | |
| /users | | | | | |
| HTTP Method | **Mapping** | **Description** | **Request Parameters** | **Response Model** | **Requires Authorization** |
| POST | /register | Used to register a new user into the system. | Payload:   * username\*   (String) * password\*  (String) * email\*  (String) * name\*  (String) * lastName\*  (String) * dateOfBirth\*  (Date) * avatarUrl  (String) * description  (String) | / | ✘ |
| PATCH | /edit-profile | Used to edit information about the user | Payload:   * name  (String) * lastName  (String) * dateOfBirth  (Date) * avatarUrl  (String) * description  (String) | / | ✔ |
| POST | /login | Used to authorize a user with the provided username and password in the request’s payload. | Payload:   * user\*   (String) * password\*  (String) | AuthorizedUser | ✘ |
| POST | /login/refresh-token | Used to request for a new access token. | Payload:   * refreshToken\*   (String) | AuthorizedUser | ✔ |
| PUT | /forgot-password | Used to mark an account as having its password forgotten to. | Payload:   * email\*   (String) | / | ✘ |
| POST | /reset-password | Used in combination with the forgot-password request to reset the password of an account. | Payload:   * secretKey\*   (String) * password\*  (String) | / | ✘ |
| GET | /skills | Returns a list of all the skills that a user has provided for themselves. | / | Skill[] | ✔ |
| PUT | /skills/{id} | Adds the skill with the ID provided in the URL path to the list of the user’s skills. | Path:   * skillId\*  (String) | / | ✔ |
| DELETE | /skills/{skillId} | Removes the skill with the ID from the user’s list of skills. | Path:   * skillId\*  (String) | / | ✔ |
| GET | /job-positions | Returns a list of the user’s preferred job positions. | / | JobPosition[] | ✔ |
| PUT | /job-positions /{id} | Adds a new preferred job position to the user’s list of job positions. | Path:   * id\*  (String) | / | ✔ |
| DELETE | /job-positions /{id} | Removes a job position from the user’s list of job positions. | Path:   * id\*  (String) | / | ✔ |
| POST | /employment-info | Adds a new preferred job position to the user’s list of job positions. | Payload:   * companyName\*  (String) * jobPosition\*  (String) * startedAd\*  (Date) | / | ✔ |
| DELETE | /employment-info | Removes employment information for the logged-in user. | / | / | ✔ |
| GET | /education | Returns a list of the logged-in user’s education info. | / | EducationInfo[] | ✔ |
| POST | /education | Adds an item to the list of the logged-in user’s educations. | Payload:   * schoolName\*  (String) * startedAt\*  (Date) * endedAt  (Date) * description  (String) | / | ✔ |
| PATCH | /education /{educationId} | Edits the education item with the ID from the list of the logged-in user’s education list. | Path:   * educationId\*  (String)   Payload:   * schoolName  (String) * startedAt  (Date) * endedAt  (Date) * description  (String) | / | ✔ |
| DELETE | /education /{educationId} | Removes the education item with the ID from the user’s education list. | Path:   * educationId\*  (String) | / | ✔ |
| GET | /certificates | Returns a list of the logged-in user’s skill certificates. | / | SkillCertificate[] | ✔ |
| POST | /certificates | Adds a new skill certificate to the logged-in user’s list | Payload:   * issuedBy\*  (String) * issuedAt\*  (String) * description  (String) * documentUrl  (String) | / | ✔ |
| DELETE | /certificates /{certificateId} | Deletes the certificate entity with the ID from the user’s certificate list. | Path:   * certificateId\*  (String) | / | ✔ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Job Position Controller | | | | | |
| /job-positions | | | | | |
| HTTP Method | **Mapping** | **Description** | **Request Parameters** | **Response Model** | **Requires Authorization** |
| GET |  | Retrieves a list of all the job positions in the system. | / | JobPosition[] | ✘ |
| POST |  | Used to add a new job position to the system. | Payload:   * name\*  (String) | / | ✔ |
| DELETE | /{positionId} | Used to delete a job position from the system. | Path:   * positionId\*  (String) | / | ✔ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Company (Organization) Controller | | | | | |
| /companies | | | | | |
| HTTP Method | **Mapping** | **Description** | **Request Parameters** | **Response Model** | **Requires Authorization** |
| POST | /register | Used to register a new company representative into the system. | Payload:   * username\*   (String) * password\*  (String) * email\*  (String) * company\*  (String) * avatarUrl  (String) | / | ✘ |
| POST | /login | Used to authorize a company user with the provided username and password in the request’s payload. | Payload:   * user\*   (String) * password\*  (String) | AuthorizedUser | ✘ |
| POST | /login/refresh-token | Used to request for a new access token. | Payload:   * refreshToken\*   (String) | AuthorizedUser | ✔ |
| PUT | /forgot-password | Used to mark a company’s account as having its password forgotten. | Payload:   * email\*   (String) | / | ✘ |
| POST | /reset-password | Used in combination with the forgot-password request to reset the password of a company’s account. | Payload:   * secretKey\*   (String) * password\*  (String) | / | ✘ |
| GET | /job-posts | Returns a Page of the logged-in enterprise user’s job positions. | / | Employment-Suggestion[] | ✔ |
| POST | /job-posts | Used to publish a new Job Listing | Payload:   * jobPositionId\*  (String) * salary  (Long) * description\*  (String) | / | ✔ |
| DELETE | /job-posts /{id} | Used to delete the existing job post with the ID in the path of the URL | Path:   * id\*  (String) | / | ✔ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Posts Controller | | | | | |
| /posts | | | | | |
| HTTP Method | **Mapping** | **Description** | **Request Parameters** | **Response Model** | **Requires Authorization** |
| GET |  | Returns a page of the latest posts. | Query:   * pageSize (Integer,  default 10) * pageIndex (Integer, default 0) | PostInfo [] | ✘ |
| GET | /{postId} | Returns a detailed version of the post with the ID provided in the URL path. | Path:   * postId\* (String) | Post | ✘ |
| POST |  | Creates a new post in the system. | Payload:   * title\*   (String) * content\*  (String) | / | ✔ |
| PATCH | /{postId} | Edits the post with the provided ID with the content in the payload. | Path:   * postId\* (String)   Payload:   * title   (String) * content  (String) | / | ✔ |
| DELETE | /{postId} | Deletes the post with the ID provided in the URL path. | Path:   * postId\*  (String) | / | ✔ |
| PUT | /{postId}/like | Toggles the existence of the post-user like relation. If relation already exists it’s deleted, it’s created otherwise. | Path:   * postId\*  (String) | / | ✔ |
| POST | /{postId}/reply | Posts a reply to the post with the ID provided in the path. | Path:   * postId\*  (String)   Payload:   * content\*  (String) | / | ✔ |
| PATCH | /{postId}/reply /{replyId} | Edits the reply to the post with the ID provided in the path. | Path:   * postId\*  (String) * replyId\*  (String)   Payload:   * content\*  (String) | / | ✔ |
| PUT | /{postId}/reply/ {replyId}/like | Toggles the existence of the reply-user like relation. If relation already exists it’s deleted, it’s created otherwise. | Path:   * postId\*  (String) * replyId\*  (String) | / | ✔ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EmployMe Controller | | | | | |
| /employMe | | | | | |
| HTTP Method | **Mapping** | **Description** | **Request Parameters** | **Response Model** | **Requires Authorization** |
| GET |  | Returns one random employment suggestion based on the user’s preferred job positions and their skillset | / | EmploymentSuggestion | ✔ |
| PUT | /{suggestionId} | Used to mark an employment suggestion as accepted, system then sends an e-mail to the Company informing them of the accepting of their employment. | Path:   * suggestionId  (String) | / | ✔ |
| DELETE | /{suggestionId} | Used to mark an employment suggestion as declined by the user, until it is updated again. | Path:   * suggestionId  (String) | / | ✔ |

### Service

The Service layer is where all the business logic should go. It performs application specific logic and manipulation, etc. The Service layer receives requests from the Controller and communicates with the Repository layers below it.

Along with the services that our system will contain, we will need to somehow inject these modules and components by always having instances of them at hand, instead of creating a new one every time we need it, which means that we will be using Dependency Injection for this, which will of course be handled by the framework of our choice – Spring Boot.

### Repository Logic

The Repository layer provides an abstraction of data, so that adding, removing, updating, and selecting items from this collection is done through a series of straightforward methods, without the need to deal with database concerns. The Repository layer will provide some form of an abstraction in our system, so that it can act as a mediator in the process of communication with the database, making sure that our queries will always be optimal, and the results will be as expected.

We will be using the repository and unit of work patterns, that are going to help us to create an abstraction layer between the data access layer and the business logic layer of an application. Implementing these patterns will help us insulate our application from changes in the data store and can facilitate automated unit testing or test-driven development.

### Object Relational Mapper

Hibernate framework will be used as ORM (Object Relational Mapper) to manage connection between API and database. It is an open-source tool that provides object-oriented approach to access relational databases. It helps developers achieve consistency and persistence in their application, using the data language and the objects developers are familiar with.

ORM is based on the containerization of objects, and the abstraction that makes it possible to address, access and manipulate objects without considering how they are related to their data sources. Hibernate is known for its excellent stability, quality, and is highly configurable and extensible. Hibernate encapsulates the use of JPA as an interface to be able to communicate with the database, acting as a translation layer between what the Services want and what the JPA will send to the database as a query, a translator of some kind.

## Database

### ER Diagram

TODO – CREATE ER DIAGRAM

# Development

## Tools

Development:

* Visual Studio Code 2022
* IntelliJ IDEA 2022.2
* NodeJS

Database Management:

* SQL Server Management Studio (SSMS)
* DataGrip 2022.1.5

## Technology stack

### Front-End

**Web Design Languages:**

* HTML5 – markup language for defining the structure of a front-end page.
* CSS5 – language for styling the front-end page.
* JavaScript - Version ES2015
* Typescript – Version 4.7.4

**Libraries**:

* React – Version 18
* React-Bootstrap - Version 2.4.0
* Material UI – Version 5.9.0

### Back-End

**Programming Languages:**

* Java – Version 17
* C# - Version 10

**Software Frameworks**:

* Spring Boot – Version 2.7.1
* .NET – Version 6.0

**Version control system**:

* Git/Azure DevOps Repos

**Database:**

* Microsoft SQL Server
* PostgreSQL

**API Documentation**:

* Swagger - We will be using Swagger that is a documentation of our API, it will help us describe the structure of our APIs so that machines can read them. This enables developers to execute and monitor the API requests they send and the results they receive, making it a great tool for developers and testers.