Bilkent University

Department of Computer Engineering

Senior Design Project

Recroute

Final Report

Ege Şahin	21702300
Göktuğ Gürbüztürk	21702383
Aybars Altınışık	21601054
Ahmet Feyzi Halaç	21703026

Supervisor: Prof. Dr. Özgür Ulusoy

Jury Members: Dr. Shervin Arashloo, Dr. Hamdi Dibeklioglu, Erhan Dolak and Tağmaç Topal

Innovation Expert: Ahmet Eren Başak

May 6, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction

Due to the digitalization of today's world and the pandemic process that has dominated the world for a while, most things have started to be done through online platforms. For these reasons, many transactions such as job interviews and tests measuring technical skills were conducted face-to-face in the past, now carried out through various digital platforms. However, there is no platform to handle all aspects of this process such as meetings between company and the applicant or tests that the applicant must complete before being accepted by the companies.

Normally, recruiters use different platforms at different stages in this process. They find the candidates from a platform, test these candidates with the help of other platforms and meet with those candidates from other online platforms. The fact that all these stages are carried out on different platforms and regulated manually creates a serious workload for recruiters. In order to reduce this problem and help companies, we aim to make all these stages easier and faster on a single platform named Recroute and reduce the workload of recruiters.

2 Requirements Details

2.1 Functional Requirements

- Recruiters will be able to create application forms by combining necessary questions (experience, skills etc.)
- Recruiters will be able to generate URL links for their application forms and share it with potential applicants via a communication channel.
- Recruiters will be able to send an invitation email to an individual from his/her LinkedIn profile page via Recroute Chrome extension.
- Recruiters will be able to create proficiency tests by combining open-ended, single choice and coding questions. There will be a question pool where recruiters can choose necessary questions among them to create tests.
- Recruiters will be able to create Zoom meetings for interviews with the applicant.

- Recruiters will be able to create recruitment flows for their job advert. These flows will consist of an application form, proficiency tests and interviews.
- Recruitment flows can be automated like rejecting all applicants without proper experience or passing all applicants with a specific skill to the next process.
- Recruiters will be able to see all applicants in a tabular format.
- Recruiters will be able to filter and search by any attribute in tables.
- There will be an action button whose purpose is passing selected applicants to the next process in the recruitment flow. If the next process is a proficiency test, it will send mail to the applicant with a link to the test. If the next process is an interview, the application will send an email for arranging a Zoom meeting.
- Applicants will be able to fill application forms on their own, or they can apply with their LinkedIn accounts or resumes to prepopulate possible fields of the form.
- Recruiters will be able to see applicants' performance on coding tests such as code quality, run time, memory usage etc.
- Recruiters will be able to see which applicants are rejected or passed in which process.

2.2 Nonfunctional Requirements

2.2.1 Performance

- Application should be quick and responsive in order to provide a user friendly experience. Clients should be able to access and send data to the server in less than 1 second response time.

2.2.2 Scalability

- Application should be able to handle 1,000,000 users simultaneously without delays as the platform is designed to be a social platform that will be heavily used by clients.

- Number of application servers should be able to increase if there is a heavy load on a server.

2.2.3 Availability

- Application should be available to use %98 of the time in a month because it will handle most of the recruitment process of a company.

2.2.4 Extensibility

- Main idea behind the application is to improve user experience over the recruitment process, thus, the application should be open to upgrades and changes according to the user feedback.

2.2.5 Privacy and Security

- Application should provide an information of applicant to a recruiter only if applicant applies for an advert created by that recruiter.
- Application should keep sensitive data encrypted.
- Application should use TLS v1.1 or higher version to provide a secure communication between client and server.
- Sandbox development environment (Coding question processor service) should be safe in order to block incoming attacks from applicants. Sandbox environment should not have administrative permissions and it needs to run on another physical machine than the main application server for any incoming attacks.

2.3 Pseudo Requirements

2.3.1 Implementation Requirements

- Git will be used for the version control system and corresponding repositories will be contained in Github.
- React.js library will be mainly used for frontend development. Implementation with React.js environment consist of Javascript, HTML and CSS usage.
- Backend development will be on Java Spring Framework.
- Chrome Extension API will be used to implement extension for LinkedIn

2.3.2 Economic Requirements

- The application will be free to use for applicants.
- Application will be software-as-a-service(SaaS) for companies thus it will be a subscription based platform.
- APIs such as LinkedIn API and Zoom API are free to use.
- Rest of the development tools are also available for free usage.

2.3.3 Sustainability Requirements

- According to user feedback, modifications on the application should be done to meet user requirements.
- As the application will be SaaS, it should be able to handle multiple requests from a large number of users without a delay.

2.3.4 Time Requirements

- A Fully functioning application should be implemented early before the CSFair 2022 to emulate the user experience on the system and enhance the application as it needs.

3 Final Architecture and Design Details

3.1 Overview

Below is the explanation of the final software architecture of Recroute. First the subsystem decomposition is explained with a diagram. Then hardware software mapping is explained with a deployment diagram. After that low level class and object diagrams are provided for each subsystem of Recroute.

3.2 Subsystem Decomposition

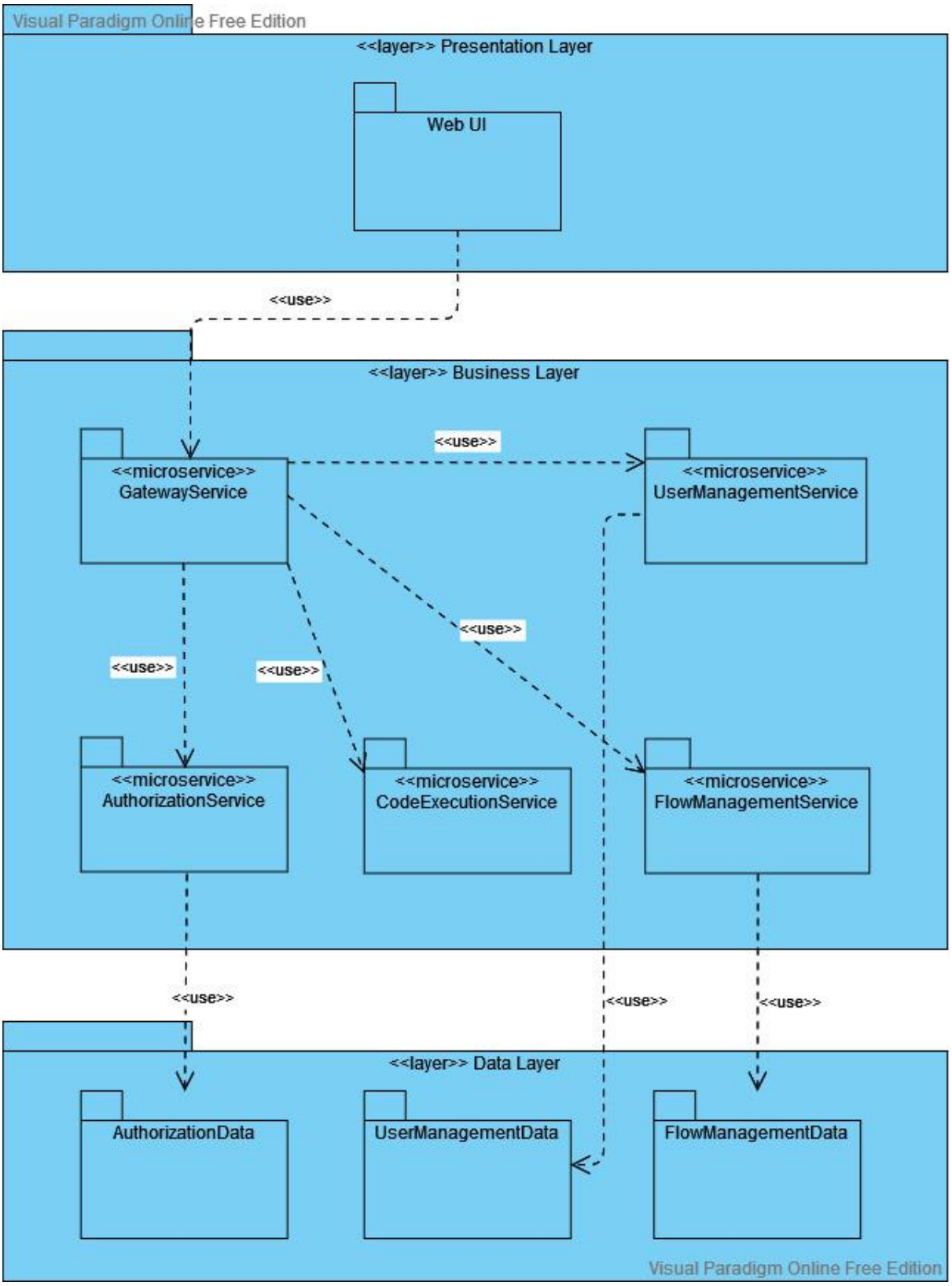

Figure 1. Recroute Subsystem Decomposition

3.3 Deployment Diagram

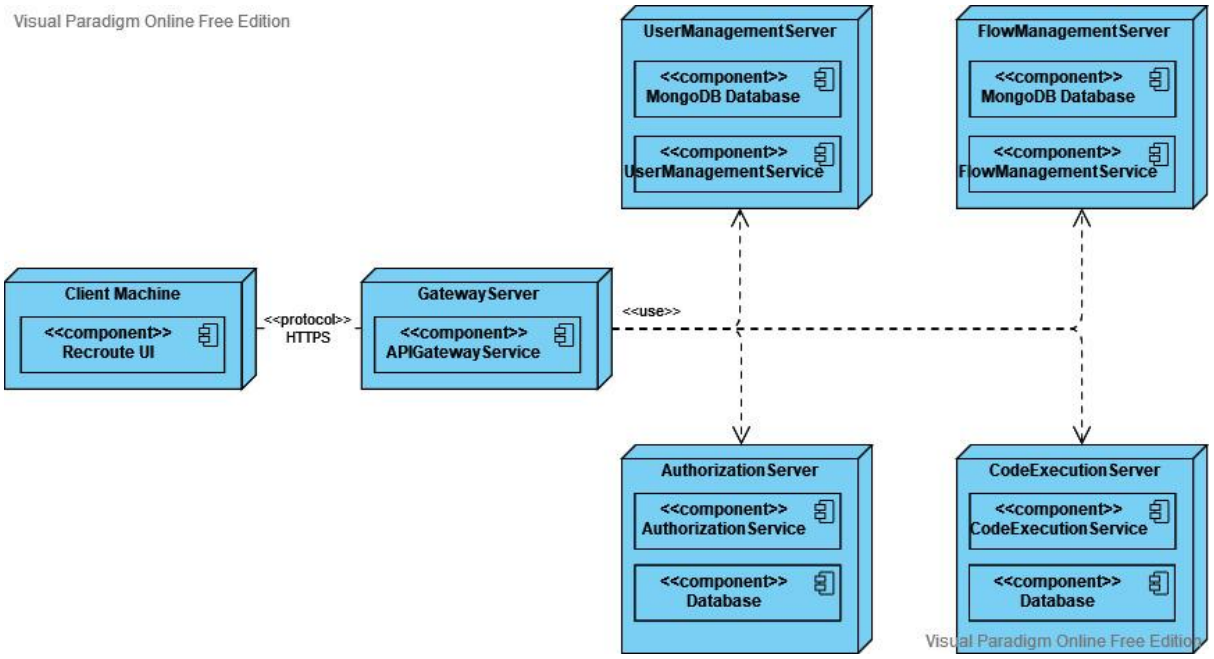

Figure 2. Deployment Diagram

Above is the deployment diagram of the final system. Each microservice is able to be deployed on a different machine to meet the scalability quality attribute of the system. Scalability improves as the number of server instances increases. As the system is designed to use the microservices architecture pattern, each module is independent. Different from the proposed system, microservices do not use a common database in the final system, instead, each microservice accesses an individual database.

3.4 Class Interfaces

3.4.1 User Management Service

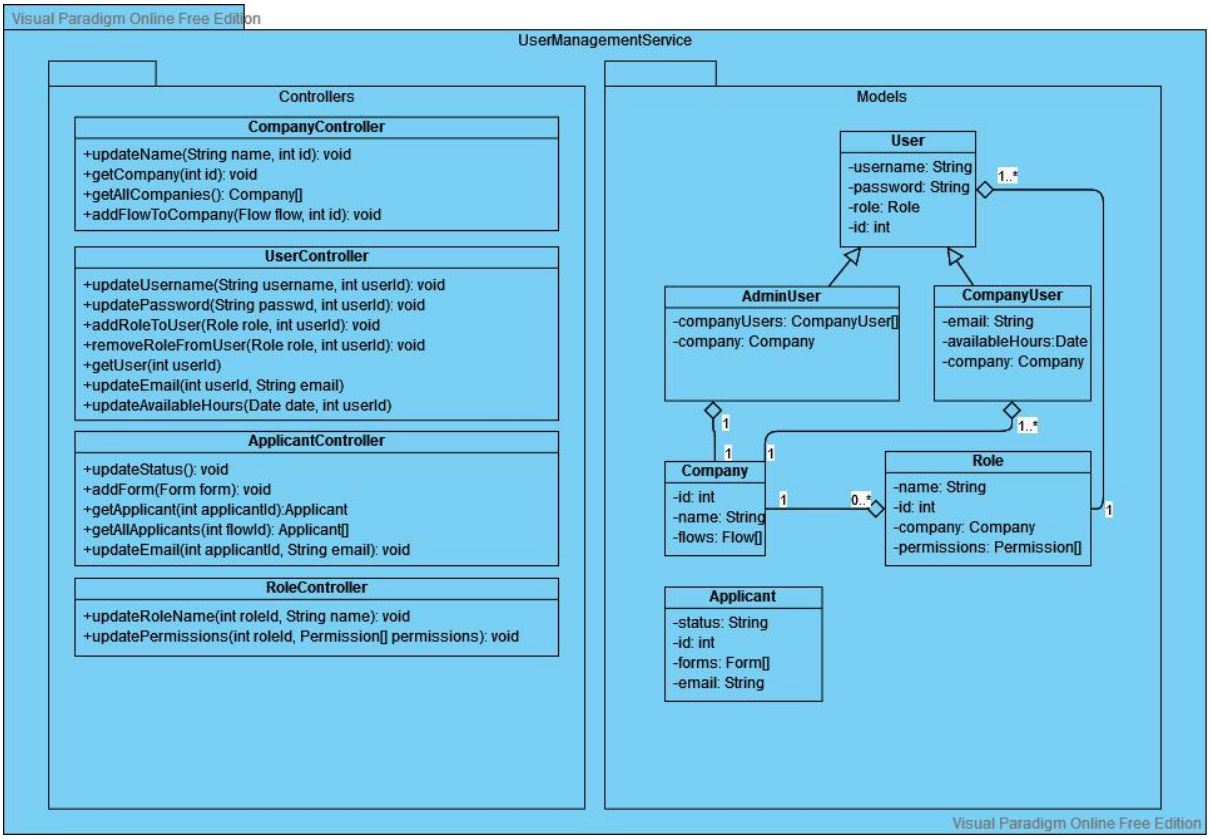

Figure 3. User Management Service

3.4.1.1 Models

- **Class:** User

  User class represents the system's users who can be either AdminUser or CompanyUser.

  Attributes:

  - ❖ userName: String
  - ❖ password: String
  - ❖ role: Role
  - ❖ id: int

- **Class:** CompanyUser

  CompanyUser class represents the recruiters that can use functionalities offered

by the application.

Attributes:

- email: String
- availableHours: Date
- company: Company
- id: int

- **Class:** AdminUser

  AdminUser is the user that controls CompanyUsers and can also reach the company related information in the system.

  Attributes:

  - name: String
  - companyUsers: CompanyUser[]
  - id: int
  - company: Company

- **Class:** Company

  Each company in the system will be an instance of this class.

  Attributes:

  - id: int
  - name: String
  - flows: Flow[]

  Class: Role

  Role class represents the role of the user in the system. It is used for authorization.

  Attributes:

  - name: String
  - id: int
  - company: Company
  - permissions: Permission[]

- **Class:** Applicant

  Each applicant in the system will be an instance of this class.

Attributes:

- status: String
- id: int
- forms: Form[]
- email: String

3.4.1.2 Controllers

- **Class:** CompanyController

  CompanyController class includes necessary functions to manipulate a company object.

  Functions:

  - updateName(String name, int id): Updates the name of the company.
  - getCompany(int id): Returns the company with a given id.
  - getAllCompanies(): Returns all the companies registered in the system.
  - addFlowToCompany(Flow flow, int id): Adds new flow to the company's flows.

- **Class:** UserController

  UserController class includes the necessary functions to manipulate a user object.

  Functions:

  - updateUsername(String username, int userId): Updates the username of the user with a given id.
  - updatePassword(String password, int userId): Updates the password of the user with a given id.
  - addRoleToUser(Role role, int userId): Adds new role to the user.
  - removeRoleFromUser(Role role, int userId): Removes role of the user.
  - getUser(int userId): Returns the user with a given id.
  - updateEmail(int userId, String email): Updates the email of the user with a given id.
  - updateAvailableHours(Date date, int userId): Updates the available time slots of the user with a given id.

- **Class:** ApplicantController

ApplicantController class includes the necessary functions to manipulate a user object.

Functions:

- updateStatus(String newStatus): Updates the status of the applicant.
- addForm(Form form): Adds a new form to the user's forms.
- getApplicant(int applicantId): Returns the applicant with a given id.
- getAllApplicants(int flowId): Returns all the applicants in the flow with a given id.
- updateEmail(int applicantId, String email): Updates the email of the applicant.

- **Class:** RoleController

  RoleController class includes the necessary functions to manipulate a role object.

  Functions:

  - updateRoleName(int roleId, String name): Updates role name.
  - updatePermissions(int roleId, Permission[] permissions): Updates the permission of a role.

3.4.2 Flow Management Service

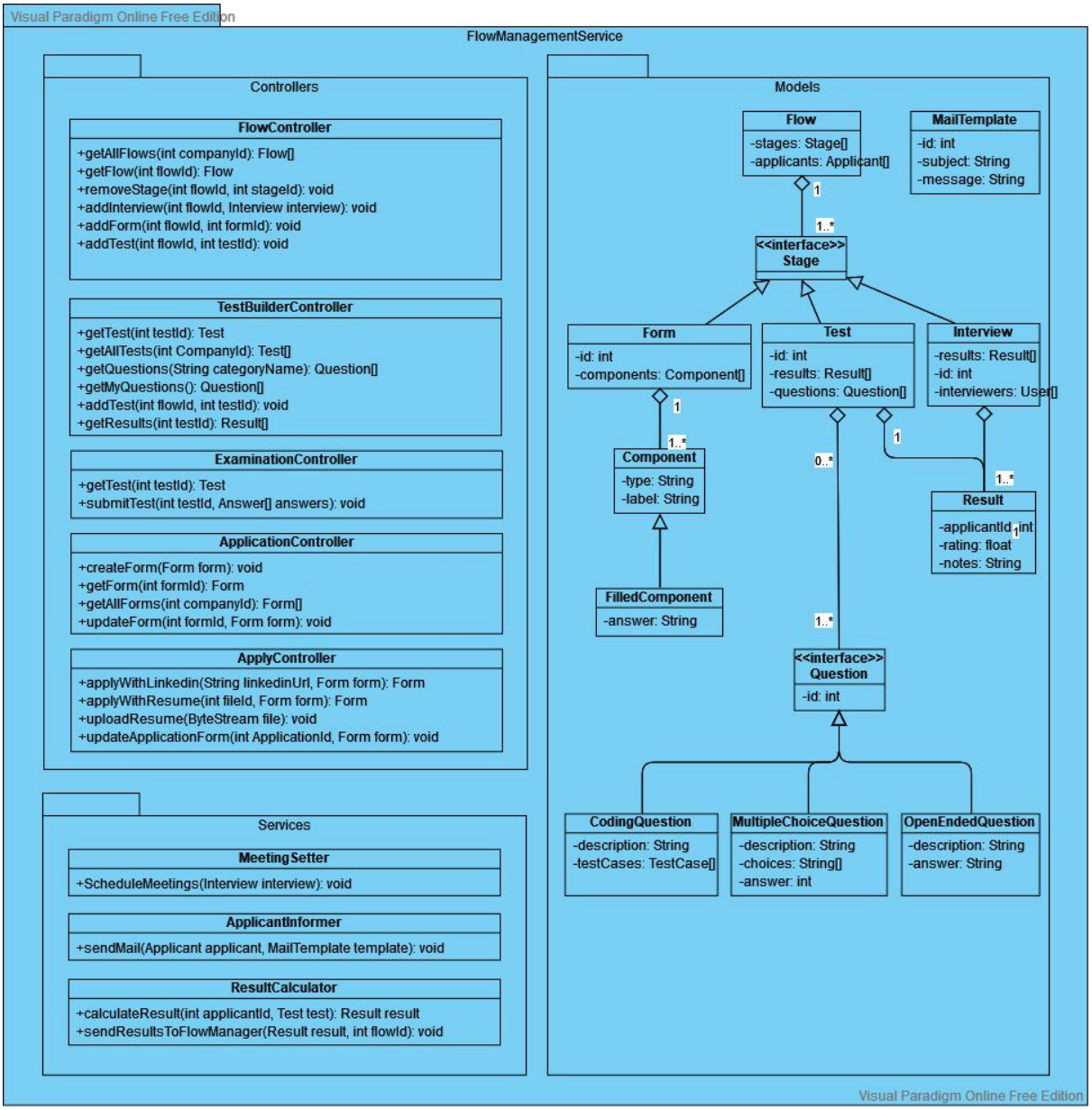

Figure 4. Flow Management Service

3.4.2.1 Models

- **Interface:** Stage

  Form, Test and Interview classes are extended from this Stage interface.

- **Class:** Flow

  Flow class represents the flow of a specific job advert. It consists of stages.

  Attributes:

  - stages: Stage[]

- applicants: Applicant[]

- **Class:** Form

  Form class represents the form that will be filled by the applicant.

  Attribute:

  - formComponents: Component[]
  - id: int

- **Class:** Component

  Component class represents the components used in the Form object. (Such as a name field, etc.)

  Attribute:

  - type: String
  - label: String

- **Class:** FilledComponent

  FilledComponent is a type of component. It inherits the Component class.

  Attribute:

  - answer: String

- **Class:** Test

  Test class represents the test which includes one or more questions and created by

  the company.

  Attributes:

  - id: int
  - results: Result[]
  - questions: Question[]

- **Interface:** Question

  Question interface represents the question created by the company or a question that is added to the system by developers. CodingQuestion, MultipleChoiceQuestion and OpenEndedQuestion extend this interface.

  Attribute:

- id: int

- **Class:** CodingQuestion
  CodingQuestion class represents the type of a question which includes coding exercise. This class extends the question interface.
  Attributes:
  - description: String
  - testCases: TestCase[]

- **Class:** MultipleChoiceQuestion
  MultipleChoiceQuestion class represents the type of a question which has multiple
  choices. This class extends the question interface.
  Attributes:
  - description: String
  - choices: String[]
  - answer: int

- **Class:** OpenEndedQuestion
  OpenEndedQuestion class represents the type of a question which is open-ended. This class extends the question interface.
  Attributes:
  - description: String
  - answer: String

- **Class:** Interview
  Interview class represents the interview stage in the flow.
  Attributes:
  - results: Result[]
  - id: int
  - interviewers: CompanyUser[]

- **Class:** MailTemplate
  MailTemplate represents a template mail that will be sent to applicants.

Attributes:

- id: int
- subject: String
- message: String

- **Class:** Result

  Result objects combine the results of the interview and test that applicant attended. Test result and interview notes are different attributes of this class.

  Attribute:

  - applicantId: int
  - rating: float
  - notes: String

3.4.2.2 Controllers

- **Class:** FlowController

  FlowController class includes the necessary functions to manipulate a Flow object.

  Functions:

  - getAllFlows(int companyId): Returns all the flows of the company with a given id.
  - getFlow(int flowId): Returns the flow with a given id.
  - removeStage(int flowId, int stageId): Removes the stage from the flow with a given id.
  - addInterview(int flowId, int formId): Adds an interview to the flow with a given id.
  - addForm(int flowId, int formId): Adds a form to the flow with a given id.
  - addTest(int flowId, int testId): Adds a test to the flow with a given id.

- **Class:** TestBuilderController

  TestBuilderController class includes the necessary functions to edit the Test object while the company is creating a test.

  Functions:

  - getTest(int testId): Returns the test with a specified test id.

- ❖ getAllTests(int companyId): returns all the tests of the company with a specified company id.
- ❖ getQuestions(String categoryName): Returns all the questions in the specified category.
- ❖ getMyQuestions(): Returns all the questions of the company.
- ❖ getResults(int testId): Returns all the results related to the specified test id.

- **Class:** ExaminationController

  ExaminationController class includes the necessary functions to submit the test
  solved by an applicant.

  Functions:

  - ❖ getTest(int testId): Returns the test with a specified id.
  - ❖ submitTest(int testId, Answer[] answer): Applicant submits the tests with given answers.

- **Class:** ApplicationController

  ApplicationController class includes the necessary functions to manipulate an application form object by the company.

  Functions:

  - ❖ createForm(Form form): Creates a new application form with a given form
  - ❖ getForm(int formId): Returns the form with a specified id
  - ❖ getAllForms(int companyId): Returns all the forms belong to the company.
  - ❖ updateForm(int formId, Form form): Updates the specified form.

- **Class:** ApplyController

  ApplyController class includes the necessary functions to manipulate an application form object by the applicant.

  Functions:

  - ❖ applyWithLinkedin(String linkedinUrl, Form form): Calls the fillWithLinkedIn function in the services package

- ❖ applyWithResume(int field, Form form): Calls the fillWithResume function in the services package.
- ❖ uploadResume(ByteStream file): Uploads resume file to the system.
- ❖ updateApplicationForm(int ApplicationId, Form form): Updates the specified application form of the specified applicant.

3.4.2.3 Services

- **Class:** MeetingSetter

  MeetingSetter class is responsible for creating a meeting schedule with respect to available time slots of CompanyUser instances'. The method of this class is used by FlowController class's methods.

  Functions:

  - ❖ ScheduleMeetings(Interview interview): Schedules a new meeting for a given interview information which includes the interviewers' (CompanyUser class instance) information (availableHours attribute).

- **Class:** ApplicantInformer

  ApplicantInformer class includes functions to manipulate models. It's function is used
  in controller classes.

  Functions:

  - ❖ sendMail(Applicant applicant, MailTemplate template): Send the information regarding the scheduled meeting.

- **Class:** ResultsCalculator

  ResultsCalculator class is used to calculate each applicant's result from the test and then send these results as a whole to examination service controllers.

  Functions:

  - ❖ calculateResult(int applicantId, Test test): Returns the test's result of the specified applicant.
  - ❖ sendResultsToFlowManager(Result result, int flowId): Sends the result of the exam to the flow manager.

3.4.3 Gateway Service

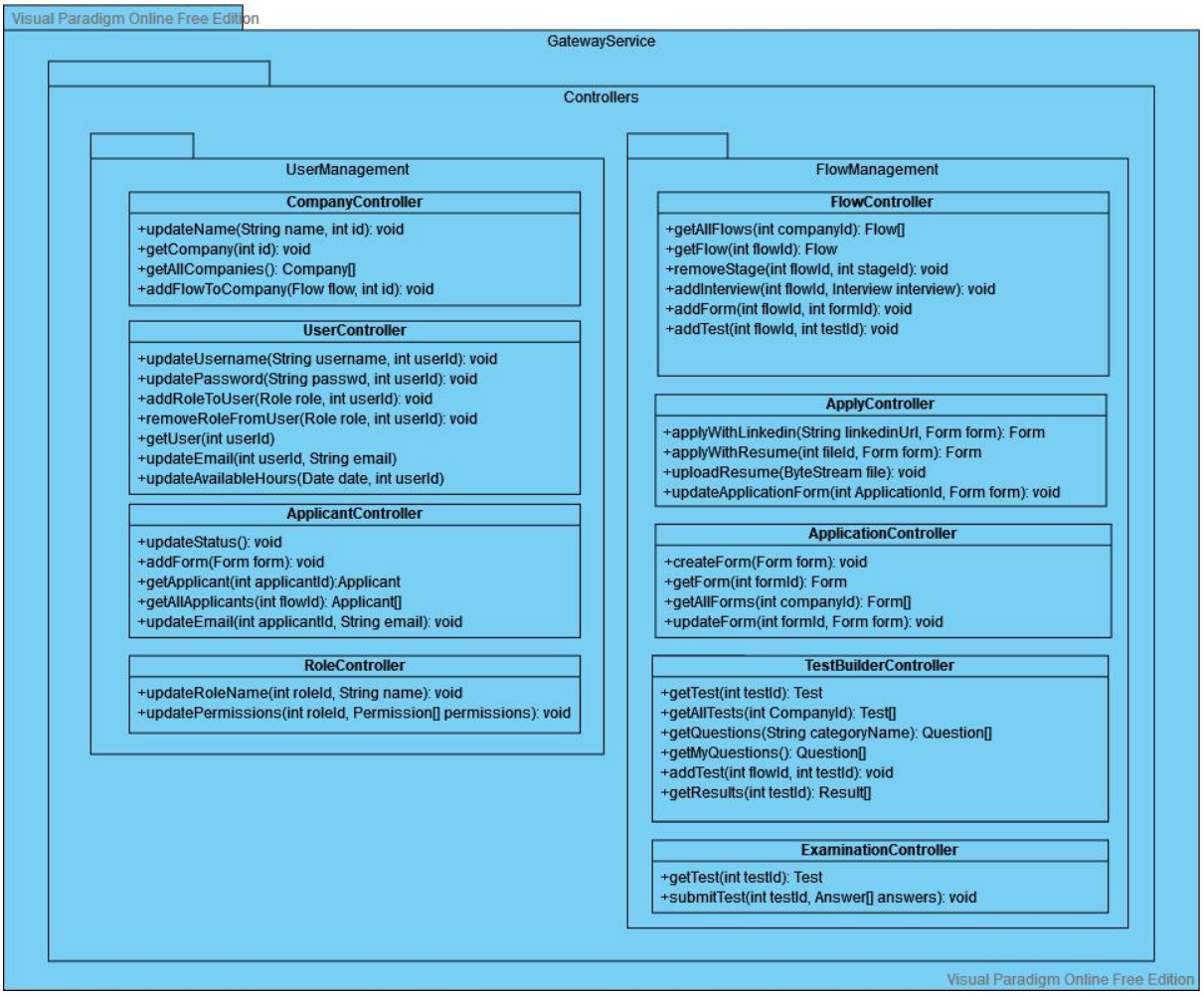

Figure 5. Gateway Service

Gateway service is responsible for routing the requests to the rest of the services. Using API gateway is very useful in microservices architecture because it provides benefits to both user and developer such as:

- Defines a better API.
- Services may change over time and should be hidden from the clients.
- Services may use different communication protocols internally.
- Limits the access to services from outside for better security.
- Load balancer may be added to the system easily.
- Number of service instances may change over time which should be hidden from the clients.

4 Development/Implementation Details

4.1 Development Details

4.1.1 Backend

We used Node.js for our microservices to be able to work with JavaScript on both frontend and backend [3]. In this way, we reduced the time to learn the used languages since multiple languages would have multiple learning curves. Moreover, along with Node.js, we used TypeScript over JavaScript because of the stability it offers compared to JavaScript.

After running the microservices, we realized that we need to rerun the service on every code change, which is very cumbersome while developing the application. So, after some research, we found a npm package called ts-node-dev [4], which can watch file changes and rerun the node process on every change. It has TypeScript support also, which fits our needs. For the production builds, we used ‘transpile-only’ option of ts-node-dev since production builds don’t need type checking because of the performance constraints.

After handling how to run the microservices we built, we looked for a solution to run multiple microservices in parallel with one command. At this point, we found another npm package called concurrently [5], which can run multiple node processes in parallel. Therefore, we mount every service on different port (starting from 3500, we increment port number by one for every service) and run them in parallel with the help of concurrently.

For deployment purposes, since we have a GoDaddy server, we used rsync protocol to deploy our local files to the production server. Since there is a running node process along with ts-node-dev, we didn’t need to do anything other than deploying our up-to-date files.

For debugging purposes, we thought that our microservices should use the production service if that service is not mounted on the local computer. So, to be able to achieve this behavior, we initialized environment variables on the parent process (concurrently) which is responsible for running services. For example, if user service is mounted on local, we initialized an environment variable called

USER_RUNNING and used this variable in our code to determine whether user service should be used from local or production.

4.1.2 Frontend

Explaining the frontend implementation details with bullet points will be more clear at this point. Therefore,

- For the JavaScript framework, we used React.js [6] since it is the most popular JavaScript framework currently. Along with React.js, we used TypeScript as a programming language because it's the strongly typed version of JavaScript, which increases code maintainability while scaling the application. Moreover, we used Next.js [7], which is a React framework which offers Server Side Rendering and lots of optimizations on top of React.
- For the global state management, we used Redux Toolkit [8], which is an extended and simpler version of Redux.
- For UI frameworks, we used both Elastic UI [9] and Material UI [10] since combination of two offers the best UI components as we see.
- For the authentication, we used Auth0 since it is the most popular authentication platform in the JavaScript community. We searched for a tool to wrap our Next.js application with Auth0 and found Next.js SDK for Auth0 [11]. With the help of this SDK, we are able to restrict the specific pages only to authenticated users.

Lastly, for debugging purposes, we thought that frontend application that is running on local server in development mode should use the backend microservices from local if that microservice is running on local, or from production server otherwise. So, after some investigation, we found a command called lsof [12], which can list the running processes along with the port they are running at. Therefore, before starting the frontend development process, we checked the ports starting from 3500 and initialized environment variables about which microservices are running on local. After that, we used these environment variables in our code to determine where to send HTTP requests.

4.1.3 Database

For the persistent storage source, we thought that using a NoSQL database will be beneficial for us since most of the data we need to store fits the BSON format more, rather than the strict SQL columns. So, we learned and used MongoDB for our database.

4.2 Folder Structure

Our project is divided into two folders as frontend and backend.

4.2.1 Frontend

4.2.1.1 Components

This folder consists of different react components that are used in different parts of the project. Components are divided into different folders with respect to which page of the project they are used

4.2.1.2 Pages

This folder consists of the react components which correspond to the screens of our application. The components from the components folder described above are used in these pages.
In addition to those, the pages folder also includes a subfolder which is Api folder. This folder includes function definitions to reach the backend of the project from the frontend.

4.2.1.3 Redux

This folder includes the global state of the application. There is a slices folder in redux folder and a store file. In the slices folder there is a file defined for each page of the application to store the page related information.
The files inside the slices folder are merged inside the store file which is again in the redux folder.

4.2.1.4 Types

In this types folder defined and these types are used in several parts in different parts of the project. These types match with the types defined in the backend.

4.2.2 Backend

4.2.2.1 Services

User and flow services are defined in the services subfolder of the backend. These services are defined in two different subfolders as “User” and “Flow”. Each of them includes their own controllers, mappers, models, routes and services as subfolders.

Services subfolder includes the services provided by the specific service (user or flow).

Models subfolder includes the necessary type declarations that are used by the specific service. These type definitions are in match with the type definitions of frontend.

4.2.2.2 Common

There is a subfolder of the backend folder as common. In this folder the mail and zoom integration of our project is managed.

4.3 Bug Solving

We wrote tests to ensure the functional requirements and improved these tests as the project develops. We used these tests for detecting bugs. After bugs are detected, we solved them by adjusting the related parts of the code.

5 Testing Details

During the early development of the application, we mainly used manual testing since it is the most time-efficient solution in the short term. Towards the end of the project, we started writing tests for the functional requirements that are important for

the main flow of the application. For explaining the technologies and approaches we used for testing, we can divide the application into two parts: API Endpoints and Frontend.

5.1 API Endpoints

For both testing and documentation purposes, we used Swagger [13], which is an API documentation tool. With the help of this technology, we were able to test the endpoints just by specifying the necessary fields such as parameters, body, etc. Other alternatives like Postman would be cumbersome since the necessary HTTP request should be constructed from scratch every time. An example UI for the Swagger documentation can be found below.

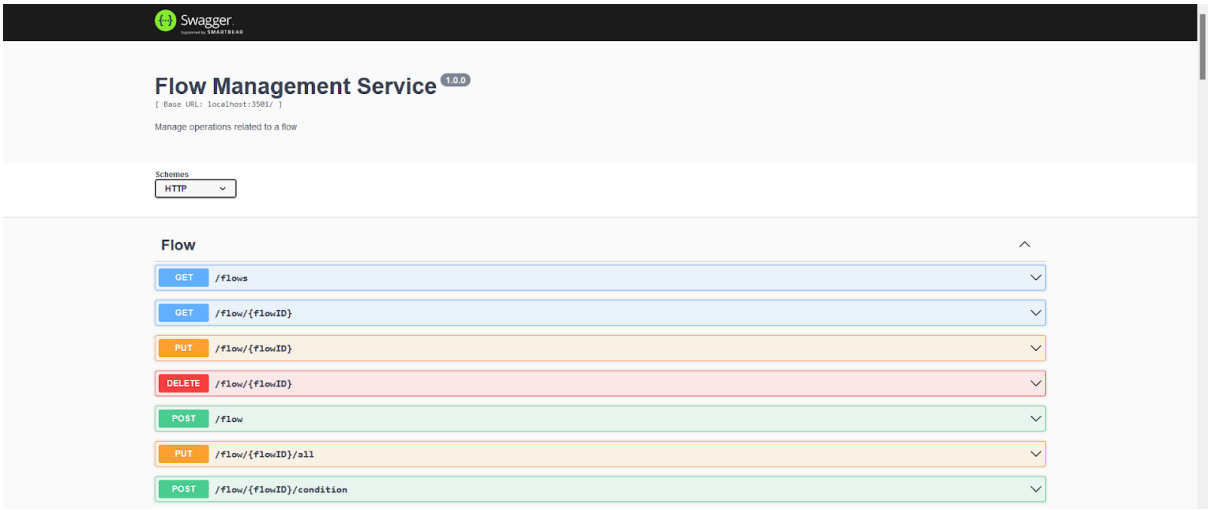

Figure 6. Main page for the Swagger documentation tool

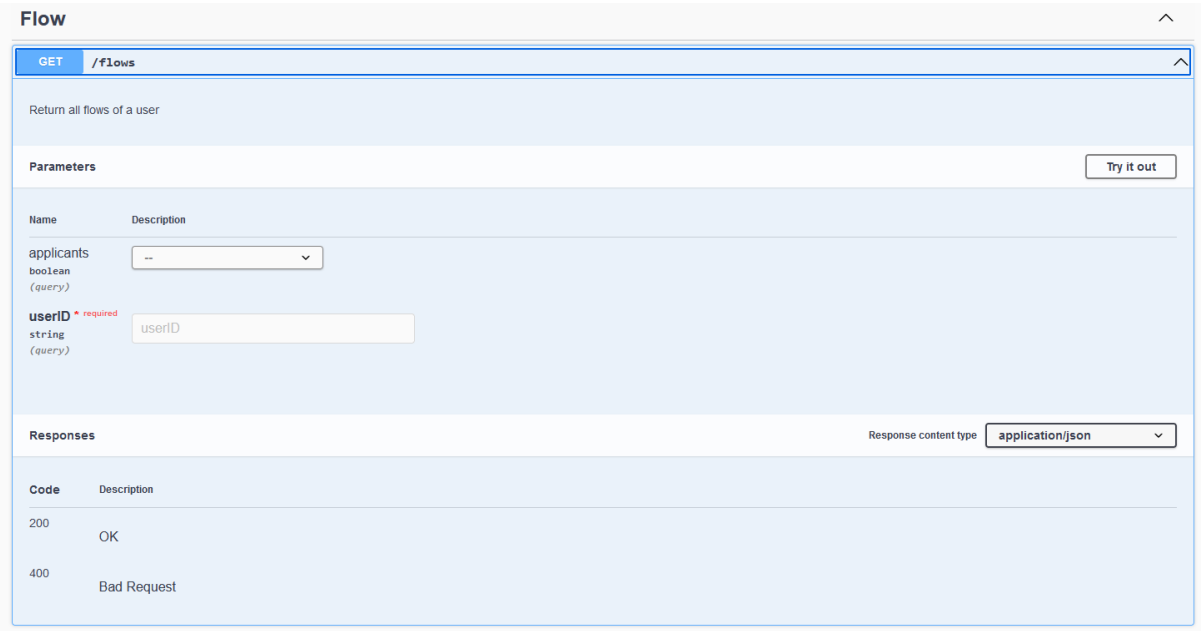

Figure 7. Explanation of a single endpoint

5.2 Frontend

For testing the integrity of our application's frontend, we wrote end-to-end tests for each functional requirement. Cypress [14] is the main technology we used while writing end-to-end tests. It is an end-to-end testing framework for web applications and bundles several web testing technologies into a single framework to offer an easy-to-install testing process. It uses Mocha for behavior-driven development like syntax and Chai for making assertions. An example test can be found below.

```
4   describe('Flows page', () => {
5       before(() => {
6           cy.loginAndVisit('/flows');
7       })
8
9       it('can see all flows', () => {
10          cy.getByTestID('flows-loading').should('not.exist');
11
12          cy.fixture('flows').then(flows => {
13              cy.getByTestID('flowList').within(() => {
14                  flows.forEach(flow => {
15                      cy.getByTestID(`${flow.name}-card`).should('be.visible');
16                  })
17              })
18          })
19      })
20
21      it('can create and delete a flow', () => {
22          const flowName = 'Test Flow';
23          cy.getByTestID(`${flowName}-card`).should('not.exist');
24          cy.getByTestID('create-flow-modal').should('not.exist');
25
26          // Open modal
27          cy.getByTestID('create-flow-button').click();
28          cy.getByTestID('create-flow-modal').should('be.visible');
29
30          // Create flow
31          cy.getByTestID('create-flow-modal-name').type(flowName);
32          cy.getByTestID('create-flow-modal-button').click();
33          cy.getByTestID(`${flowName}-card`).should('be.visible');
34
35          // Delete flow
36          cy.getByTestID(`${flowName}-delete`).click();
37          cy.getByTestID('approve-confirmation-button').click();
38          cy.getByTestID(`${flowName}-card`).should('not.exist');
39      })
40
41  })
```

Figure 8: Test file for flows page functional requirements.

In order to make sure every functional requirement works without a problem, we ran all the tests we wrote after every change (commit). Thanks to this approach, if the change leads to a bug in one of the functionalities, we could detect that problem at an early stage.

6 Maintenance Plan and Details

Currently, we are using a virtual private server offered by GoDaddy. We deployed our project to this server which is used as a production environment. The server has 8 CPU cores and 16 GB RAM which provides more flexibility and easier scalability even if user traffic increases. It also has 400 GB NVMe SSD storage space which ensures to keep the system sustainable in case of an increase in the number of users. It costs 496,99 TL per month [1]. We have also paid 200 TL annually to maintain domain and SSL certificates. SSL certificate is needed in order to protect the user data. According to our current maintenance plan, we expect that our server will meet the system requirements.

The Recroute uses MongoDB to store the data in the backend. We preferred to use MongoDB because it provides high maintainability by providing high performance which is beneficial for high traffic [2]. Our database is also deployed on our VPS, which helps us to detect the errors by tracking the error logs on the server. Apart from these, we are expecting that no extra costs are required in order to provide higher maintainability. If the application has too many users than the expected, we will consider upgrading the server to so as not to harm the user experience. But for now, the server capabilities will not cause a problem as long as we do maintenance and repair the server from time to time until a serious user potential emerges.

7 Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

Recroute is an online platform which combines the different recruitment steps into a single platform. Therefore, it does neither affect nor is affected by public health, public safety and public welfare in any way at all. Also environmental and economic factors are irrelevant for the design of Recroute.

Global Factors:

Recroute is imagined as an online platform to be used by companies and applicants around the world. However, people speak different languages around the globe. This is why the general interface of the Recroute will be English so that people from different countries can interact with the platform.

Cultural Factors:

In different parts of the world, information shared by the applicants through the internet might be different from other parts of the world. This is why the privacy boundaries of different cultures need to be respected while designing Recroute.

Social Factors:

In Recroute platform, while people are applying to a job advert they share personal information through the system. In designing Recroute, the security of this information must be maintained.

	Effect Level	Effect
Public Health	0	-
Public Safety	0	-
Public Welfare	0	-
Global Factors	4	Interface should be English
Cultural Factors	9	Privacy
Social Factors	9	Data protection
Environmental Factors	0	-
Economic Factors	0	-

Table 1: Factors

7.2 Ethics and Professional Responsibilities

Recroute will follow the IEEE Code of Ethics[1]. In projects where the personal data of a user is managed, privacy is an important value. In our platform, applicants will share personal data with the companies. However this data will not be stored by the platform and the shared data will not be shared with any third-party without the permission of the users.

Also companies will be notified that they must not use applicants' data for other purposes than recruitment.

The sources that will be used in the project(libraries, APIs etc.) will be used according to license agreements and they will be referenced on the documents.

7.3 Judgements and Impacts to Various Contexts

In the recruitment processes that took place before the pandemic, the stages were carried out just like the processes carried out in the online environment. In this respect, we did not have to juggle much because we thought that these processes should be carried out somehow. When we started our project, we thought that this application would be used by many companies because it benefits society in various ways, as indicated in the table below.

Judgement Description	Impact Level	Impact Description
Impact in Global Context	High	Since most of the companies have started to work remotely, the recruitment processes have been conducted online. At this point, our considerations went into how to make the recruitment process easier for both companies and applicants in the world.
Impact in Economic Context	Medium	We considered that conducting the whole process online may prevent companies from using physical resources. It will also be less costly in terms of economy due to the decrease in resource use. In this respect, we thought that it would be very beneficial for companies in economic terms.
Impact in Environmental Context	Low	The environmental context was a low priority issue while making a decision. However, our consideration showed that the application may decrease the paper usage in the recruitment processes. This indirectly contributes positively to the environment.
Impact in Societal Context	Low	Since we are storing the information about the user for the hiring company, we needed to consider what concerns candidates might have about the privacy of their information. However, we assumed that this issue would not cause a big problem, since the same concern may be present in the current online job application processes.

7.4 Teamwork Details

7.4.1 Contributing and functioning effectively on the team

As developers of the Recroute project we believe that good teamwork can be achieved with people who fulfill their responsibilities and help each other when faced with various obstacles. In order to achieve this, we held meetings at regular intervals. In these meetings, we were discussing how the problems can be eliminated and how to share the works that need to be completed at the next stage. While sharing the work, we considered the past experiences of the team members in order to enable them to function effectively. This kind of sharing minimizes the problems that may arise and prevents delaying of the project by ensuring that everyone contributes at the maximum level.

7.4.2 Helping creating a collaborative and inclusive environment

All team members helped to create a collaborative environment by joining meetings. Thanks to these meetings, we get everyone's opinion on the subject and create an inclusive environment. We used many synchronous and asynchronous communication tools for that purpose. Also, Jira is used to distribute tasks and track progress more easily. In this way, team members could see what each other was doing and could help each other when needed. During the implementation phase, we also used Github to collaborate with other team members.

7.4.3 Taking lead role and sharing leadership on the team

We prepared different work packages which are explained detailly in the Analysis Report. For each work package we selected a leader who prepares the tasks to be completed at that work and controls the progress of the work at regular intervals until the deadline. Leadership roles are distributed fairly among the team members in order to improve their leadership skills without giving all responsibility to one person.

7.4.4 Meeting objectives

To be able to meet the project objectives, we had a meeting on Sunday of every week in order to assign new tasks for the following week. Also, those meetings allowed us to follow what we were doing in the previous week. In this way, we were

able to detect the missing tasks early and take the necessary steps. At this point, we used Jira to specify the tasks and assignees. Also, we used Github as a version control system to work collaboratively.

7.5 New Knowledge Acquired and Applied

While implementing our project we tried to learn new and popular technologies. We handled our work sharing accordingly so that each member of our group could learn something new. Every member of our group acquired new knowledge and developed himself. At the end, all of us get familiar with most of the tools we used during development. While learning these tools, we mainly used stackoverflow and other related web pages.

Since we tried to create a user-friendly application, for the frontend of the project we learned TypeScript and React/Redux and improved our ability to use them. As our database we preferred to use MongoDB which is an integrated suite of cloud database services. We learned about MongoDB while developing our project.

We also have a chrome extension in our project so we tried to get familiar with the architectural styles of Chrome extensions and the utilities of the Chrome extension API.

For the testing of our project, we learned and used cypress which is a javascript testing framework. Cypress can test anything that runs in a browser so we used it for end-to-end testing.

In addition to those we learned about how to deploy a web page which is an essential part of developing web pages to publish it to the web.

8 Conclusion and Future Work

In conclusion, we designed and implemented a web application that can facilitate the work of people included in the recruitment process based on automation and

integration concepts. Recroute is an application that is up and running on the production domain http://recroute.co. The Recroute has also Chrome Extension which is officially published at Chrome Web Store. This extension allows users to send potential applicants with ease. We believe that the application is functional and usable for a company although it needs further implementation and maintenance for the future work.

We aim to continue the development process to meet described functional requirements in the requirements section. Adding new integrations to our platform is necessary for companies to be formed with their habits in the recruitment process. We are also open to new ideas and functionalities. Therefore, we are confident that our project has valuable future growth potential.

9 Glossary

- **Application form:** Form that should be filled by the applicants when applying for a job.
- **Job advert:** Overall application process created by company. It is composed of a recruitment flow and general information about the job.
- **Applicant:** Person who applies for a job advert.
- **Recruiter:** Company or a person who is responsible to hire employees.
- **Recruitment Flow:** The process of analyzing, testing and interviewing the job applicants and then finding the appropriate candidates for the job.
- **Stage:** A step on the recruitment flow. Can be one of the following: Application, Test or Interview.

10 References

[1] “Kendi Yolunuzu çizin: Godaddy TR,” *GoDaddy*. [Online]. Available: https://www.godaddy.com/tr-tr. [Accessed: 06-May-2022].

[2] “The Application Data Platform,” *MongoDB*. [Online]. Available: https://www.mongodb.com/. [Accessed: 06-May-2022].

[3] Node.js, *Node.js*. [Online]. Available: https://nodejs.org/en/. [Accessed: 06-May-2022].

[4] “TS-node-dev,” *npm*. [Online]. Available: https://www.npmjs.com/package/ts-node-dev. [Accessed: 06-May-2022].

[5] “Concurrently,” *npm*. [Online]. Available: https://www.npmjs.com/package/concurrently. [Accessed: 06-May-2022].

[6] “React – kullanıcı arayüzleri geliştirebileceğiniz bir JavaScript Kütüphanesi,” – *Kullanıcı arayüzleri geliştirebileceğiniz bir JavaScript kütüphanesi*. [Online]. Available: https://tr.reactjs.org/. [Accessed: 06-May-2022].

[7] “Next.js by vercel - the REACT framework,” *by Vercel - The React Framework*. [Online]. Available: https://nextjs.org/. [Accessed: 06-May-2022].

[8] *Redux Toolkit*. [Online]. Available: https://redux-toolkit.js.org/. [Accessed: 06-May-2022].

[9] Elastic, “Elastic/EUI: Elastic UI framework ,” *GitHub*. [Online]. Available: https://github.com/elastic/eui. [Accessed: 06-May-2022].

[10] “The React Component Library You always wanted,” *MUI*. [Online]. Available: https://mui.com/. [Accessed: 06-May-2022].

[11] auth0, “Auth0/nextjs-auth0: Next.js SDK for signing in with auth0,” *GitHub*. [Online]. Available: https://github.com/auth0/nextjs-auth0. [Accessed: 06-May-2022].

[12] “LSOF(8) - linux man page,” *lsof(8): open files - Linux man page*. [Online]. Available: https://linux.die.net/man/8/lsof. [Accessed: 06-May-2022].

[13] “API development for everyone,” *Swagger*. [Online]. Available: https://swagger.io/. [Accessed: 06-May-2022].

[14] “JavaScript end to end testing framework,” *JavaScript End to End Testing Framework | cypress.io testing tools*. [Online]. Available: https://www.cypress.io/. [Accessed: 06-May-2022].