Bilkent University

Department of Computer Engineering

Senior Design Project

Recroute

Low-Level Design 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

Feb 23, 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.

This report is the low level design report for Recroute which will show the detailed design and implementation packages for Recroute in an understandable manner. Our system models are explained through various UML diagrams. The report also contains analysis of Recroute’s object design trade-offs.

1.1 Object Design Trade-offs

Security over Performance

In the application many personal identifiable information will be used and stored. It is important to secure this information against cyber attacks. This is why security programs will be used in this application however this will reduce the performance of the application. From the project team’s perspective security is more important than the small delays caused by the security. Thus, security is prefered over performance.

Functionality over Portability

The main purpose of the application is to provide functions that will facilitate companies during recruitment. In this respect, we will use all of our resources to create a fully featured product. Adaptation to other platforms like an application for smartphones or tablets will not be available in the minimum viable product. So, functionality is preferred over portability.

Robustness over Rapid Development

The flow feature of the application is connected with all other smaller features such as creating application forms etc.. This is why it is important for the application that the flow feature operates correctly. If any of the stages in the flow malfunctions this will collapse all the flow. This is why robustness is prefered over rapid development. The necessary time will be taken to reduce the flaws of the application.

Cost over Scalability

Since there are similar web applications in the world, we will try to decrease the cost while providing additional functionalities. In the initial versions of the application, the scalability will not be as important as cost. In addition to the functions offered, we will aim to increase the number of users by providing an advantage over the products in the market in price. In this respect, we minimize the cost by decreasing server requirements. So, functionality is preferred over scalability.

1.2 Interface Documentation Guidelines

In this report following conventions are used:

- In the packages part, each service and its sub packages are shown in a diagram. The definition of these services are explained in the following part.
- In the class interfaces part, each service is divided into sub packages as models, controllers and services. The sub packages contain classes. First each service is divided into its sub packages and each class in those packages are defined as follows:
    - Class or Interface keyword is given in bold followed by class or interface name. After that, the definition of that class/interface is given.
    - Attributes and functions keywords are given in bold followed by the list of attributes or list of functions respectively.
    - Attributes and functions of each class are represented with list item symbols.
    - In the attributes list the name and type of the attributes are given.
    - In the functions list the name and parameter of the functions are given followed by the definition of that function.

1.3 Engineering Standards (e.g., UML and IEEE)

In this report IEEE referencing style was used for citations [1] and Unified Modeling Language (UML) was used to visualize the design of our system [2].

1.4 Definitions, Acronyms, and Abbreviations

UML: Unified Modeling Language

IEEE: Institute of Electrical and Electronics Engineers

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.

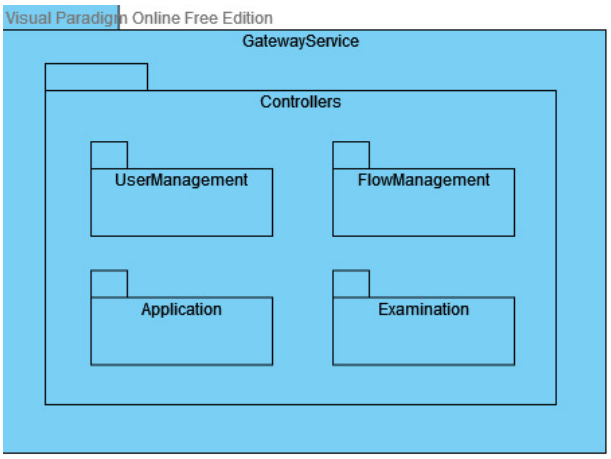
Applicant: Person who applies for a job advert.

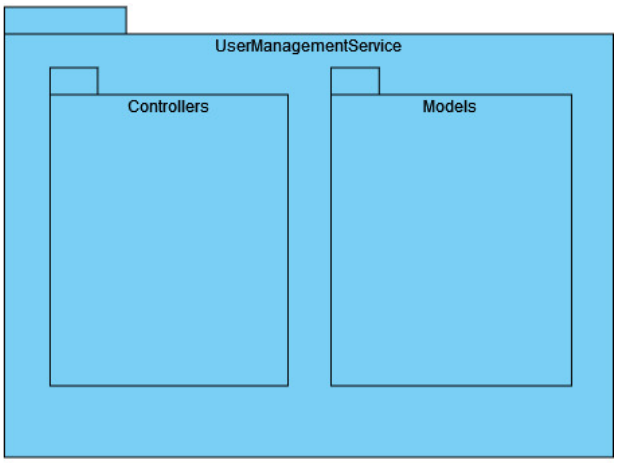
Recruiter: Company or a person who is responsible to hire employees.

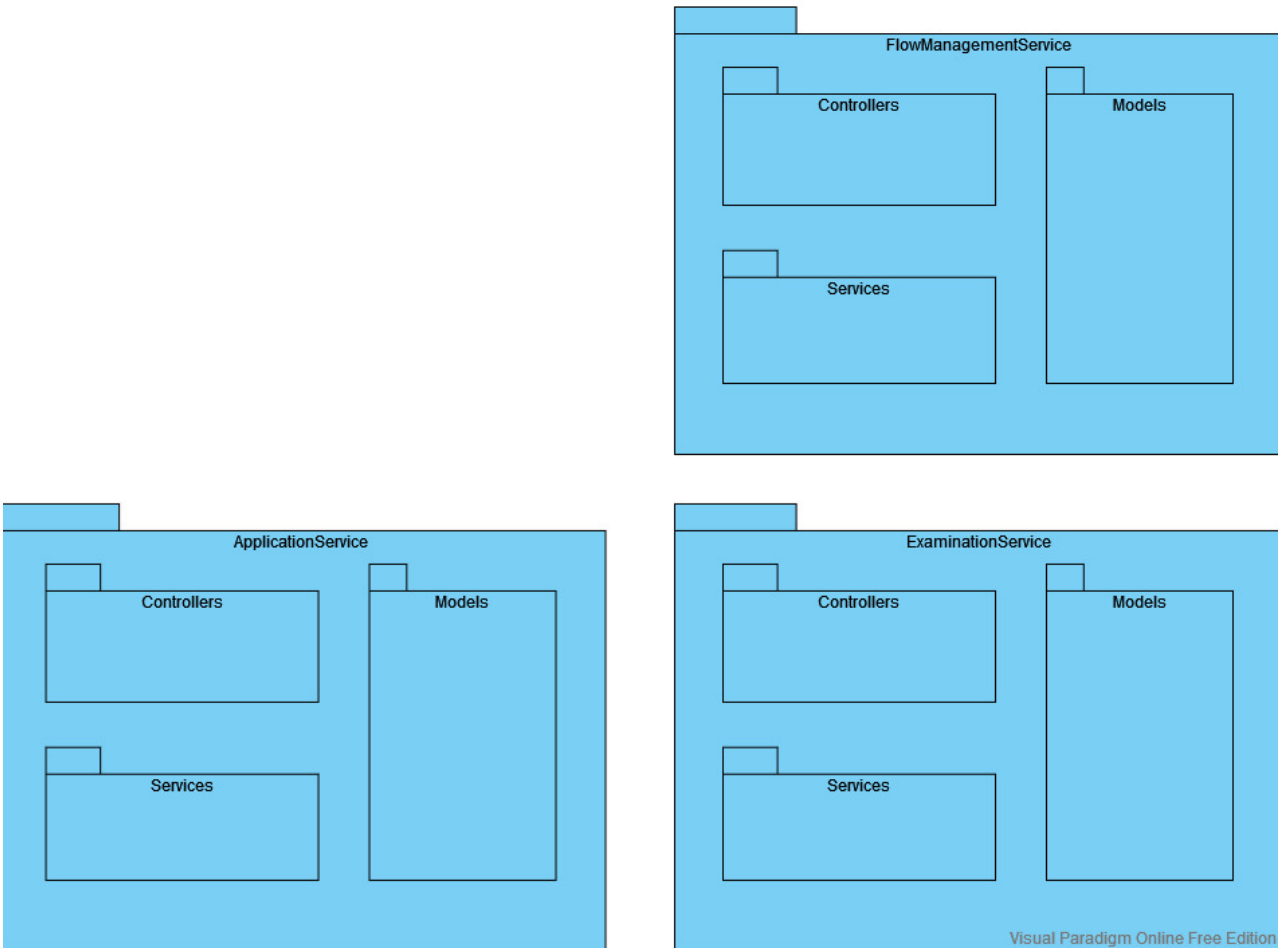
Recruitment Flow: The process of analyzing, testing and interviewing the job applicants and then finding the appropriate candidates for the job.

Stage: A step on the recruitment flow. Can be one of the following: Application, Test or Interview.

2. Packages





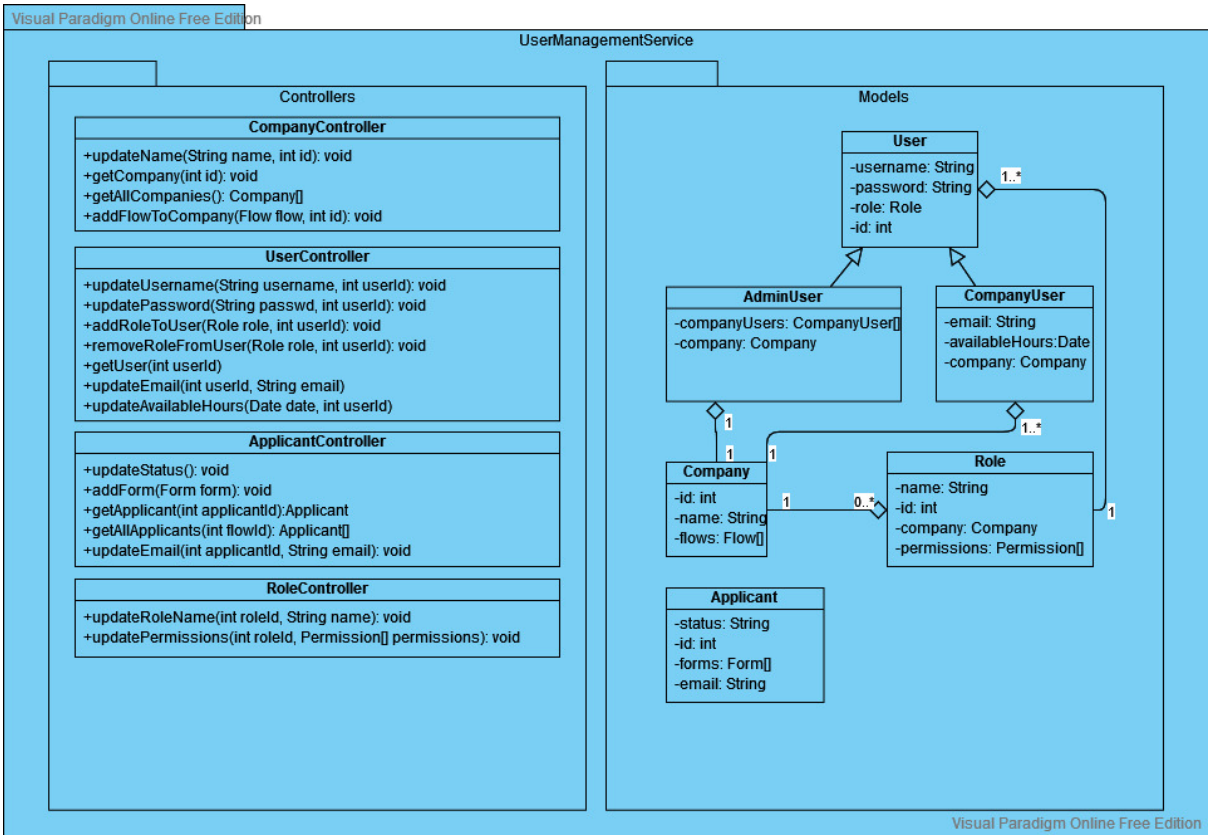


- **User management service** is responsible for the user and account operations.
- **Flow management service** is responsible for operations related to the recruitment flow. The creation of the flow and stages as well as the details of them are operated under this microservice.
- **Examination service** is responsible for the exams that recruiters will arrange. Exam management and details will be done on this service.
- **Application service** is responsible for filling application forms. An applicant may fill an application form using either a LinkedIn account or a CV. They will be implemented in this microservice.
- Each service above contains models and controllers packages. Models packages include classes for objects to be used in a specific service. Controllers packages include different functions to manipulate those objects within the particular service.
- In addition to models and controllers packages some services have services packages. The services package inside each service includes additional functions to be used by functions of the controller package.

- **Gateway service** acts as a bridge between the controllers in the other microservices. In this respect, Gateway service includes controllers of the other microservices and acts as an API.

3. Class Interfaces

3.1. User Management Service



3.1.1. 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

3.1.2. 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.

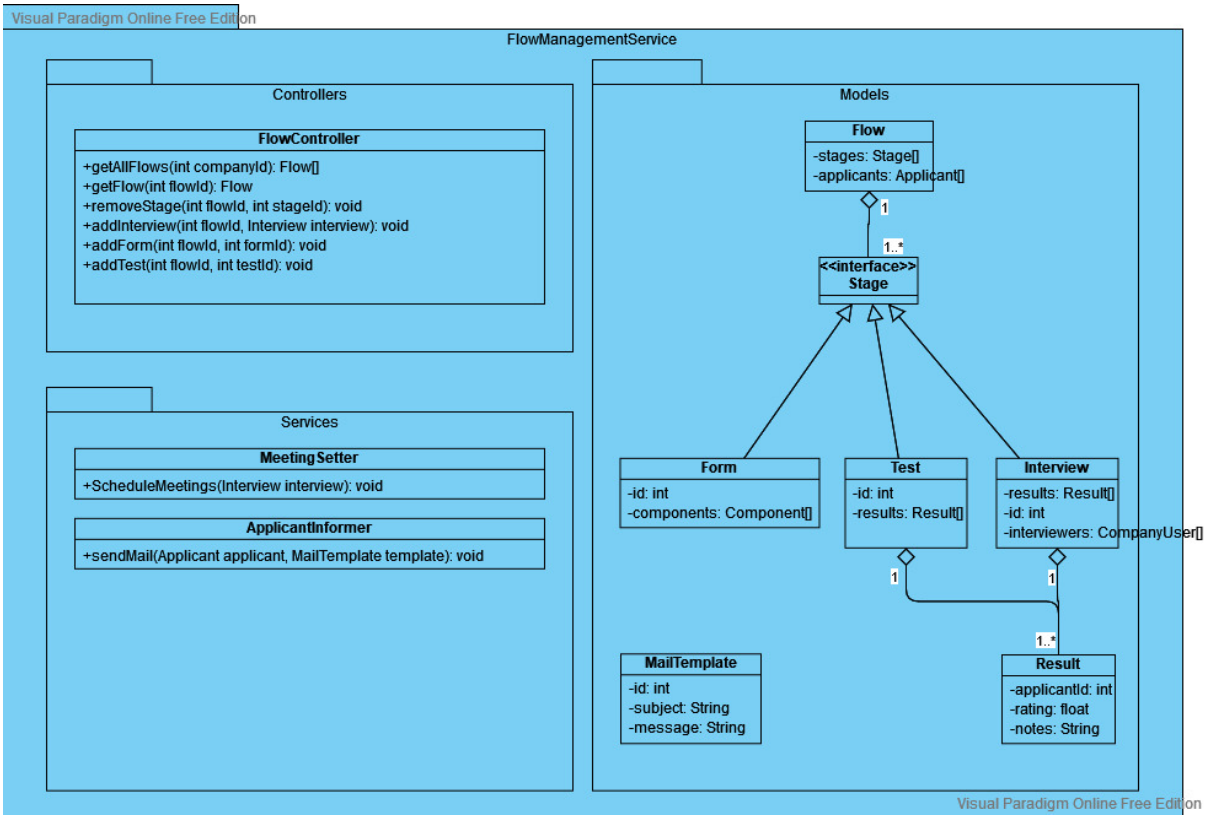
- **Class:** RoleController

  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.

3.2. Flow Management Service



3.2.1. 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:** ApplicationForm

  ApplicationForm class represents the form object created in the flow.

Attributes:

- id: int
- components: Component[]

- **Class:** Test

  Test class represents the test stage in the flow.

  Attribute:

  - id: int
  - results: Result[]

- **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

3.2.2. 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.

3.2.3. 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).

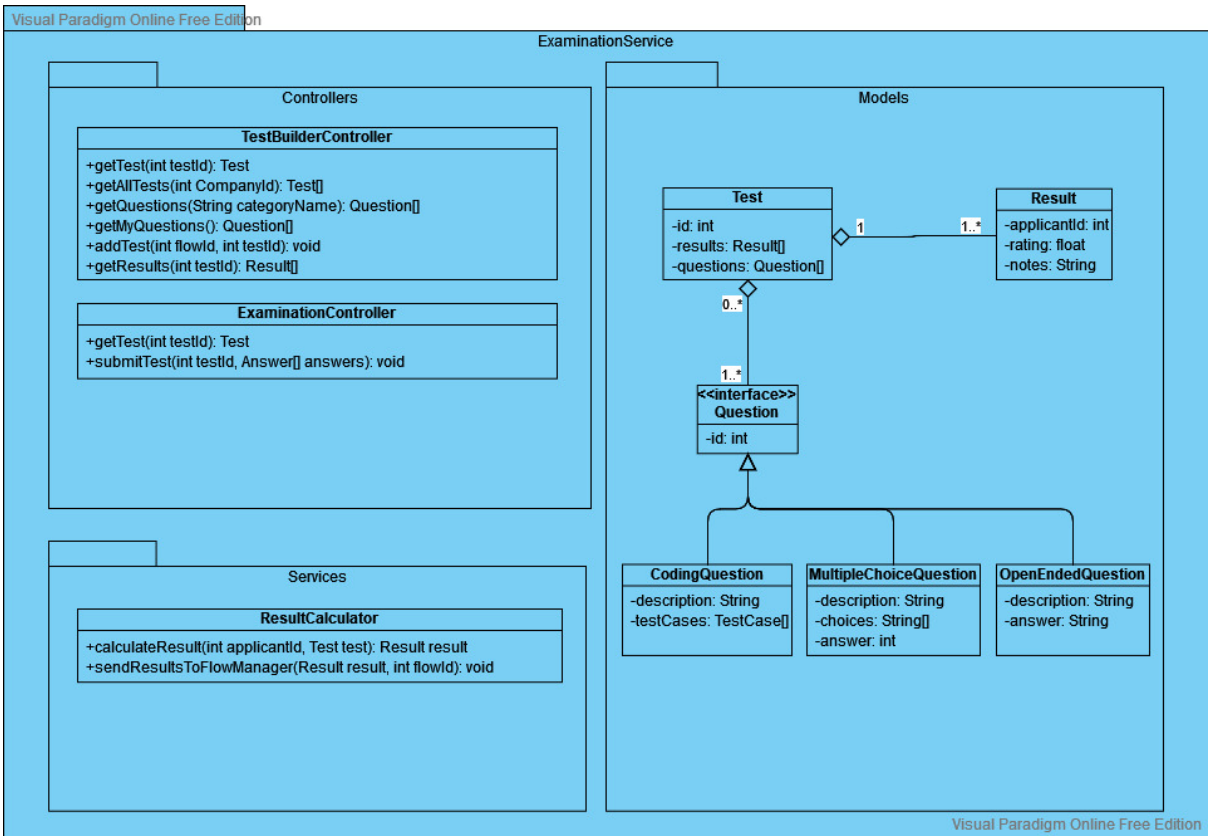
- **Class:** ApplicantInformer

  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.

3.3. Examination Service



3.3.1. Models

- **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[]

- **Class:** Result

  Result class represents the result of the test in the flow.

  Attributes:

  - applicationId: int
  - rating: float
  - notes: String

- **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

3.3.2. Controllers

- **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.

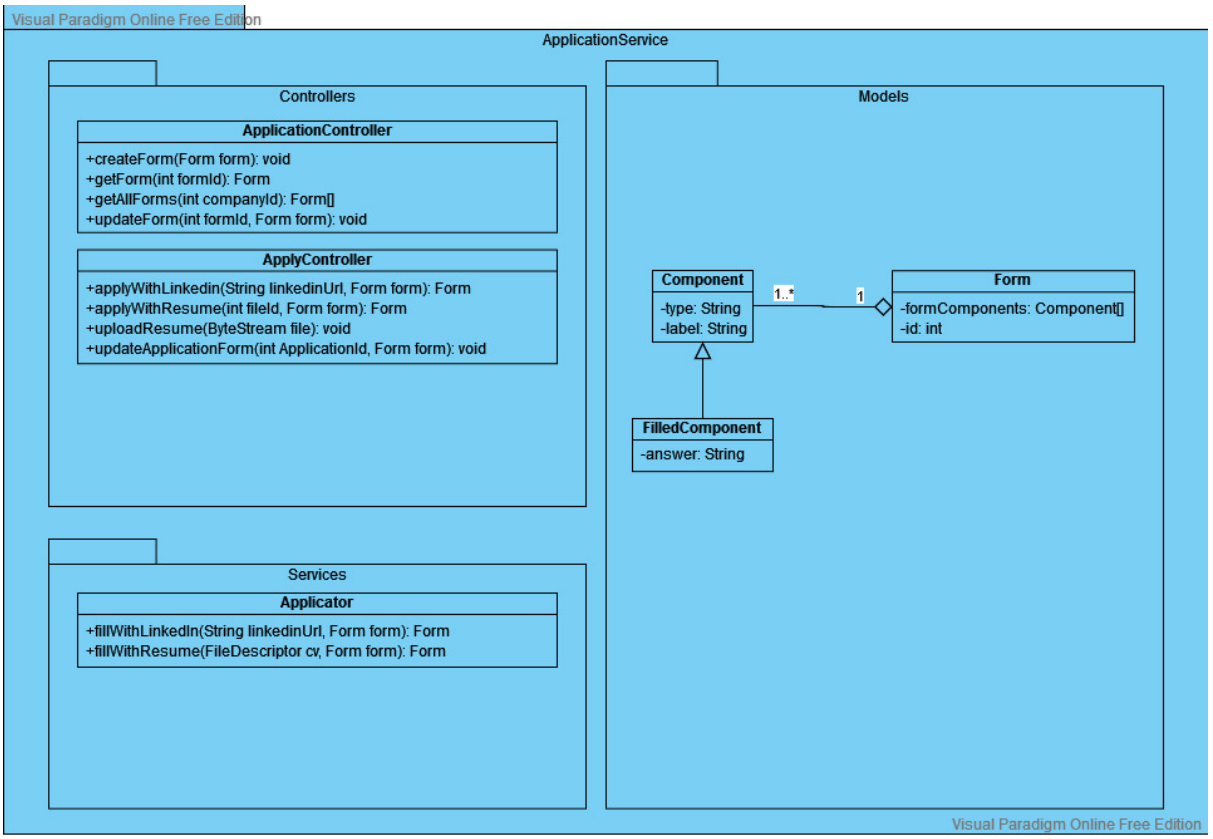
- ❖ getMyQuestions(): Returns all the questions of the company.
- ❖ getResults(int testId): Returns all the results related to the specified test id.

- **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.

3.3.3. Services

- **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. Application Service



3.4.1. Models

- **Class:** Component

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

  Attribute:

  - type: String
  - label: String

- **Class:** Form

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

  Attribute:

  - formComponents: Component[]
  - id: int

- **Class:** FilledComponent

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

  Attribute:

  - answer: String

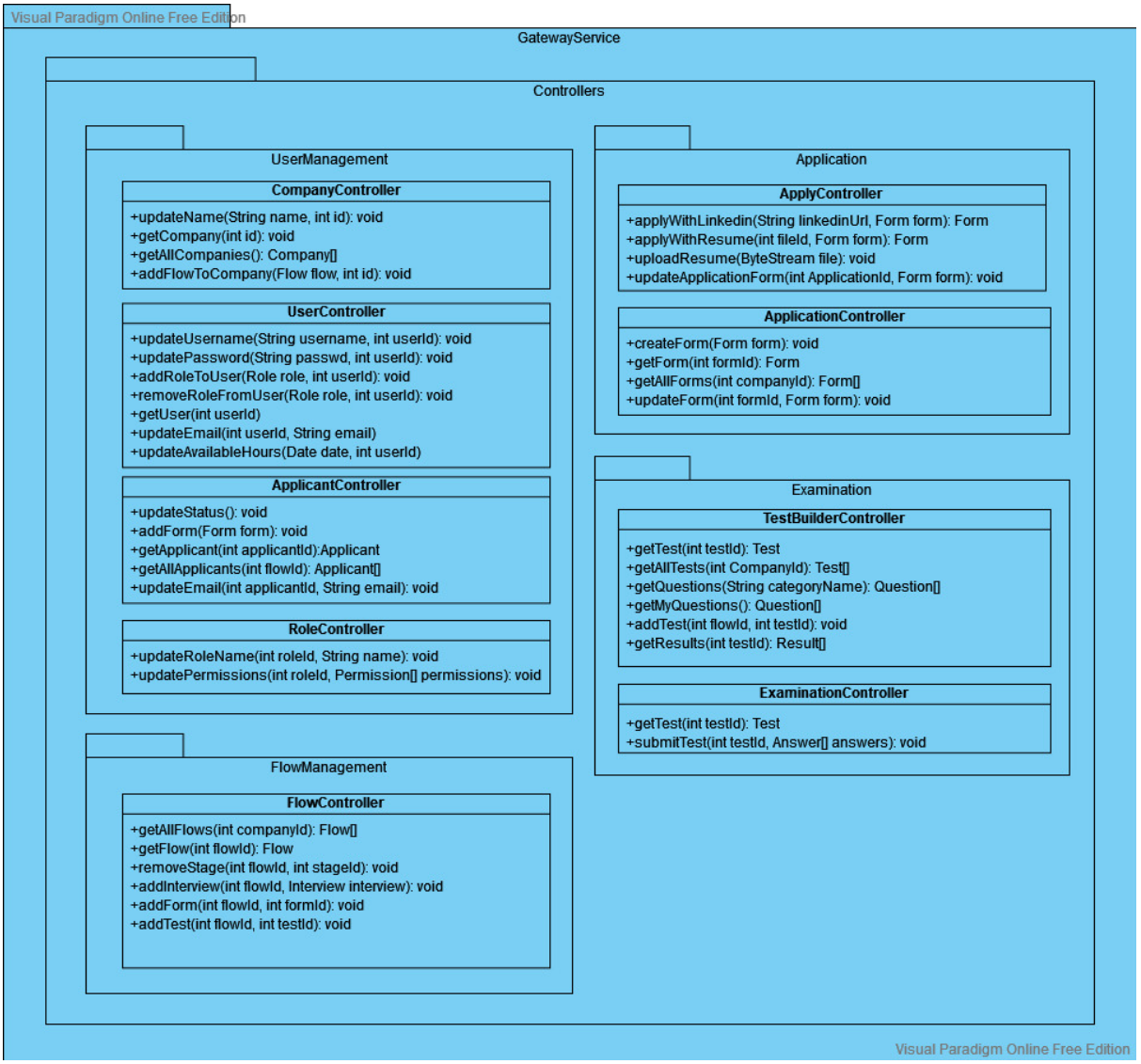
3.4.2. Controllers

- **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.

3.4.3. Services

- **Class:** Applicator
  Applicator class is used to fill the application form with the information parsed from the resume or LinkedIn account. It is used in the controller classes.
  Functions:
  - fillWithLinkedIn(String linkedinUrl, Form form): Gets the information belonging to the applicant from LinkedIn and fills the form with that information.
  - fillWithResume(FileDescriptor cv, Form form): Parses the resume and uses that information to fill the application form.

3.5 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. 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.

5. References

- [1] “IEEE Referencing: Getting started with IEEE referencing,” Library Guides. [Online]. Available: https://libraryguides.vu.edu.au/ieeereferencing/gettingstarted#:~:text=%E2%80%9CIEEE%E2%80%9D%20stands%20for%20The%20Institute,paper%2C%20provided%20in%20square%20brackets.&text=This%20is%20known%20as%20an,of%20the%20work%20are%20provided. [Accessed: 20-Feb-2022].
- [2] What is Unified Modeling Language (UML)? [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 20-Feb-2022]