Bilkent University
Department of Computer Engineering

2022-2023 Fall Semester

CS 319: Object-Oriented Software Engineering

ERASMUS PROJECT

Group 2H

DESIGN REPORT ITERATION 2

Group Members	Student Numbers
Bora Yılmaz	22003359
Cengizhan Terzioğlu	22003014
Emirkan Derköken	22002693
Göktuğ Kuşcu	22002867
Murat Güney Kemal	22002692
Onur Asım İlhan	21903375

Instructor: Eray Tüzün

Teaching Assistants: Metehan Saçakçı, Emre Sülün, Muhammad Umair Ahmed, İdil Hanhan, Mert Kara

Table of Contents

1. Introduction

1.1 Purpose of the System

The application is a web-based Erasmus and exchange application system for Bilkent University. The primary goal of the application is to provide an easy application process for students and to provide a more organized system for the instructors and coordinators. The application also aims to minimize the usage of paper and emails.

1.2. Design Goals

The design goals of the application are outlined by the non-functional requirements of the project. The application aims to provide an easy to use and reliable solution for students and instructors.

1.2.1 Usability

Since the application will be used by a wide range of students and instructors, the website will be designed with simplicity in mind to provide an easily navigable interface for all users. The design of the website will guide the user in the right direction and boundary conditions will make sure that the user does not deviate from the right procedures.

1.2.2 Reliability

The website will prevent any loss of data in the case of a system crash. Any occurring error will appropriately indicate the error's reason and consequences.

1.2.3 Performance

The website will process any operation and show its result in under 3 seconds. The loading times of the website will be in under 2 seconds. The performance will be optimized to make the website accessible for any type of computer. In order to do this Erasmus Application will do most of the hard work on the server side so any

computer with an internet connection and a browser that can render HTML5 will be able to use the website.

1.2.4 Maintainability

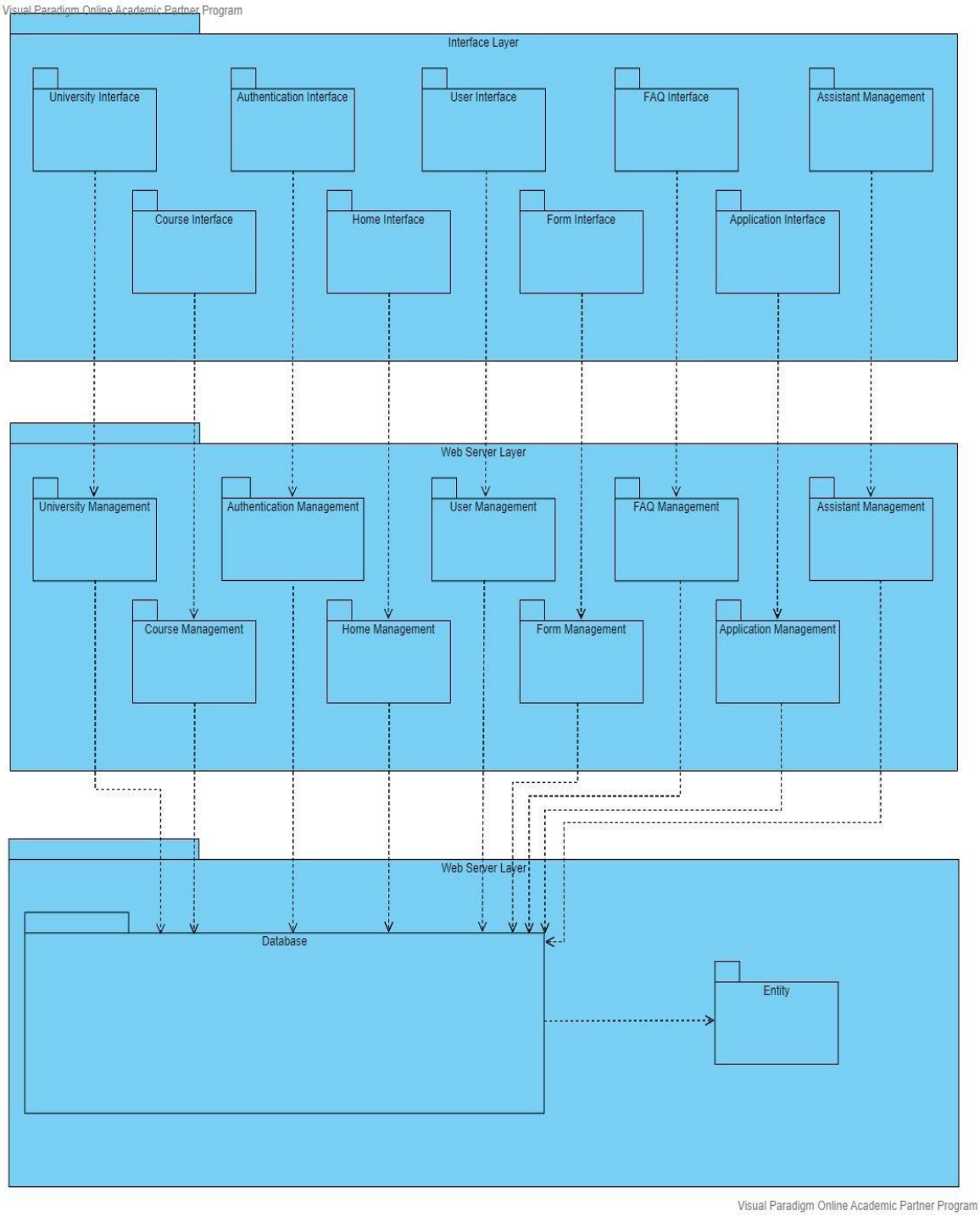
Erasmus Application will also be designed with maintainability in mind. In order to become maintainable, Erasmus Application will use OOP concepts to ease the further development processes.

1.2.5 Security

The website contains many sensitive information like the ID’s of the students. Therefore, the website will only provide the required data for each operation and will not allow any users to access sensitive information. The website’s aim is to prevent any data breaches. HTTPS protocol will be used to secure the connections of the website.

2. High-level Software Architecture

2.1 Subsystem Decomposition



A three-layer architecture is chosen in order to ensure the maintainability goal is satisfied.

Interface Layer consists of packages that will keep the boundary objects, which are the website pages. Every page depends on at least one of the controller packages from the Web Server Layer which allows user interaction through buttons that are implemented on the page. Each package includes related pages and functions those pages are able to perform. This layer is designed to ensure user-friendliness and simplicity goals are satisfied.

Web Server Layer consists of packages that will keep the controller objects. These controller objects contain functions that are performed on the server side. Each package in this layer is created to perform a different core functionality. For example, the Course Management package contains all of the functionalities that are related with courses such as course proposal, course approval, getting a list of already approved courses etc. Each package depends on the data management layer.

Data Management Layer contains Database and Entity packages. Database package is created to communicate with the database so that data addition, retrieval and deletion operations can be performed. Entity package keeps persistent objects that are created during the usage of this application.

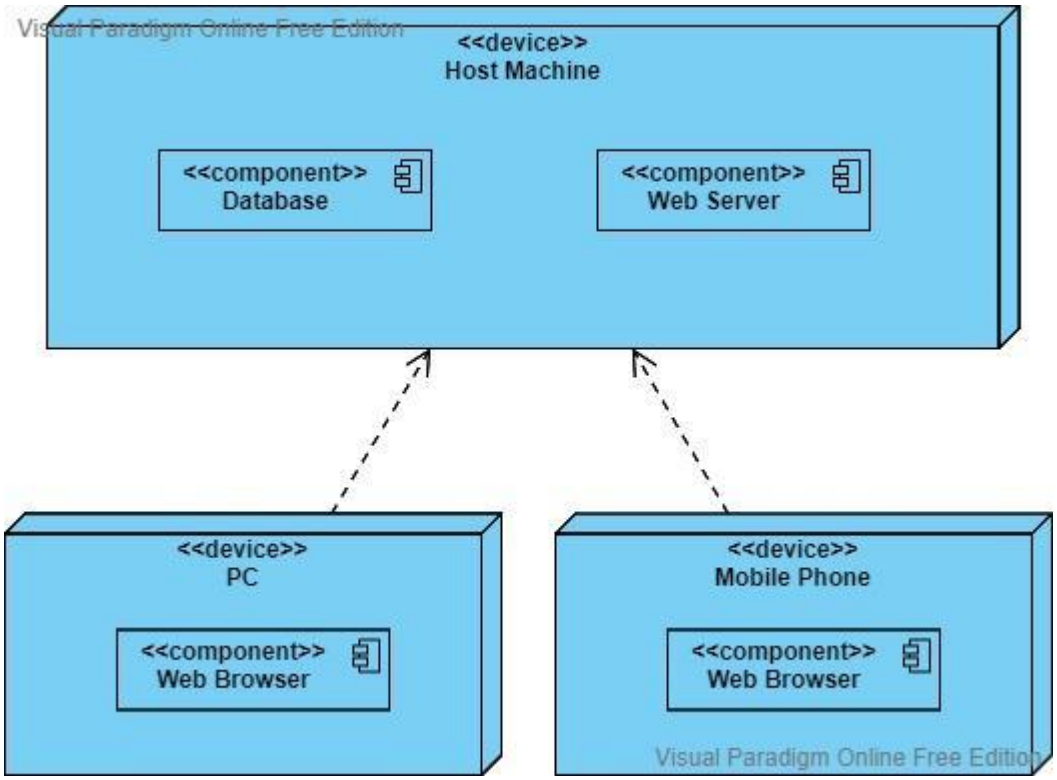
2.2 Hardware/Software Mapping

Additional hardware components aren’t needed to run our Erasmus application project. It is a web-based project so web browsers are needed and hardware systems must be strong enough to run a web browser. Web browser usage is compulsory to use our application, but it can be used on any electronic device as long as they have web browser support. For personal computers some I/O devices are necessary such as keyboard, mouse and monitor.

The website uses HTML5 and CSS3 for its frontend, and for the backend it uses Python 3.11, Flask 2.2.2, SQLite 3.

The application runs in web browsers such as Google Chrome version 108, Mozilla Firefox version 107, Safari version 16 and Opera version 93. The website might be incompatible for old web browsers such as Internet Explorer 8 due to API and external library usage.

To host the website, the server should have the minimum hardware of an Intel Pentium 4 2.8 GHz processor, 8 GB of RAM, and 128 GBs of storage. The server should also be equipped with Python 3.10 and the required libraries to run the website.



2.3 Persistent data management

We selected SQLite to use in our project. We chose SQLite because it is free and object relational which makes OOP principles to be easily applied. Also our team mostly had experience with SQLite database which was another reason to choose

it. Objects will be kept as tables and each table will consist of key-value pairs. There will be tables for Application, Course, Role, University, User and they will be persistent. In addition to this data, the PDF files generated during the Erasmus process will also be kept persistent. Data kept in the objects will be edited using the services that are implemented in the Web Server Layer and requests for editing will be done through the Database package of the Data Management Layer. The data can change at any time so it is important to send HTTP requests such as GET, POST and DELETE to the database dynamically. SQLAlchemy will be used to send these requests.

2.4 Access control and security

Access control and security are very essential concepts for our erasmus application project. Each user will have different permissions that are determined according to their roles. When the user logs in to the system, our system checks all the roles that user has and then opens the home pages related to those roles. These homepages have access to other role related pages that the user has permission to enter. Any page that the user does not have permission to will not be shown. Only the required information is pulled from the database to our front-end for each page to make the system more secure. Each password will be hashed to give extra protection.

	Student	Erasmus Coordinator	Course Coordinator	International Office	Administrative Committee
Login	x	x	x	x	x
Create Application	x				
Selection Assistant	x				
FAQ	x	x	x	x	x
Set FAQ		x			
View applications		x		x	x
Course proposal	x				
Approve course proposal		x	x		

Add course		x	x		
Update course status		x	x		
Add university		x			
Update university		x			
Decline course proposal		x	x		
To do	x	x	x	x	x
Final approval		x		x	x
Application Status	x	x		x	x
View courses	x	x		x	x
View application period	x	x	x	x	x
Update application period		x			
Upload placements				x	
Set preferred university		x			
Cancel application	x				
Upload Form	x	x		x	x
Update application	x	x			

2.5 Boundary Conditions

2.5.1 Initialization

The website will be hosted on a web server. The database will also be hosted on a web server, and the website will be connected to that database instead of a local database. To initialize the website the Python file ”run_server.py” will be executed by the developers. Users don’t need to install any program to their devices since our project runs on a web server at all times. This makes the use of the application easier. Users just need a device that has web browser support and

access to Internet connection. Before log in only log in, sign up, FAQ and partner universities pages can be seen. User needs an account that is kept in the database to be able to log in to the system. The users can perform their actions from different web browsers or different tabs of the same web browser at the same time. For each web browser, they will have to log in separately. Though they will not have to log in again to perform an action in a different tab of the same web browser if the user logged in to the system from another tab. All data will be initialized from the database at the time of the creation of the page.

2.5.2 Termination

The Erasmus Application makes use of a four layered architecture. Hence the whole system does not have to be shut down when one of the subsystems needs to be terminated for maintenance. Furthermore, the debug mode of the Flask framework makes it possible to perform maintenance while the website is up. Users will be notified at least one day prior to the termination if terminations are planned terminations. When a shutdown has to occur, first every existing data will be saved to the database to prevent any loss, then the website will be terminated.

2.5.3 Failure

When an error occurs, a message will be displayed to the users, detailing what went wrong (e.g “The course you have chosen cannot be added as it does not have a Bilkent equivalent.”). For debugging purposes, any issues regarding the developer side will be logged in detail to a file.

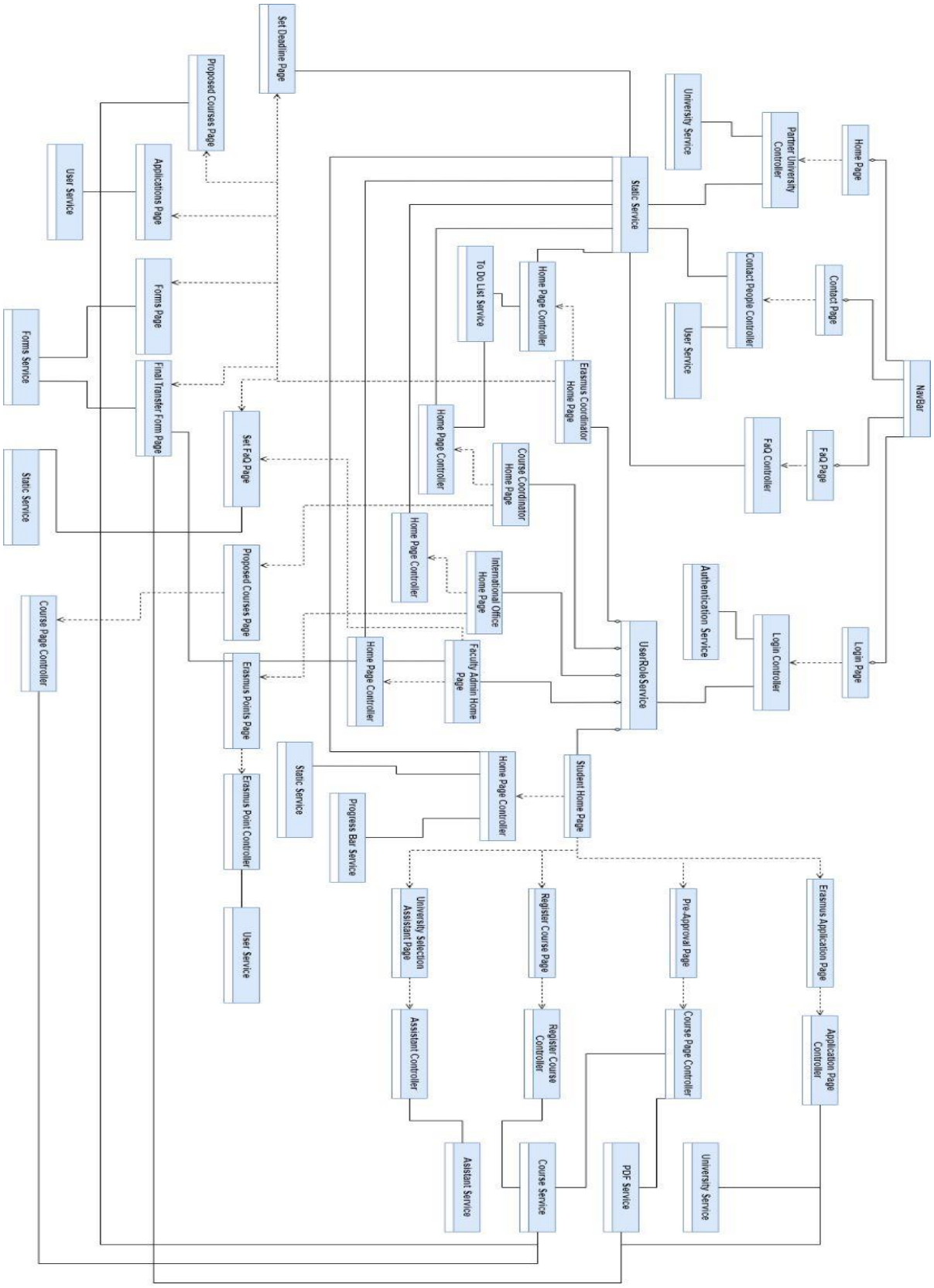
3.Low Level Design

3.1 Object Design Trade-offs

Scalability versus Production Speed: This application makes use of a templating engine in order to build and render HTML files in the backend using Python language instead of using JavaScript in the frontend. This technique greatly increases the production speed and production cost since it requires less developers and less effort. However; it also requires a lot more computation to be made on the server-side since a request has to be sent even for making slight changes on the boundaries and little to no computation is done in the users' machine. Consequently; as the website traffic gets hotter, the performance drops more significantly compared to an API approach. However; since this application is meant for public use and there is a challenging time constraint, it is decided that production speed is more important than scalability

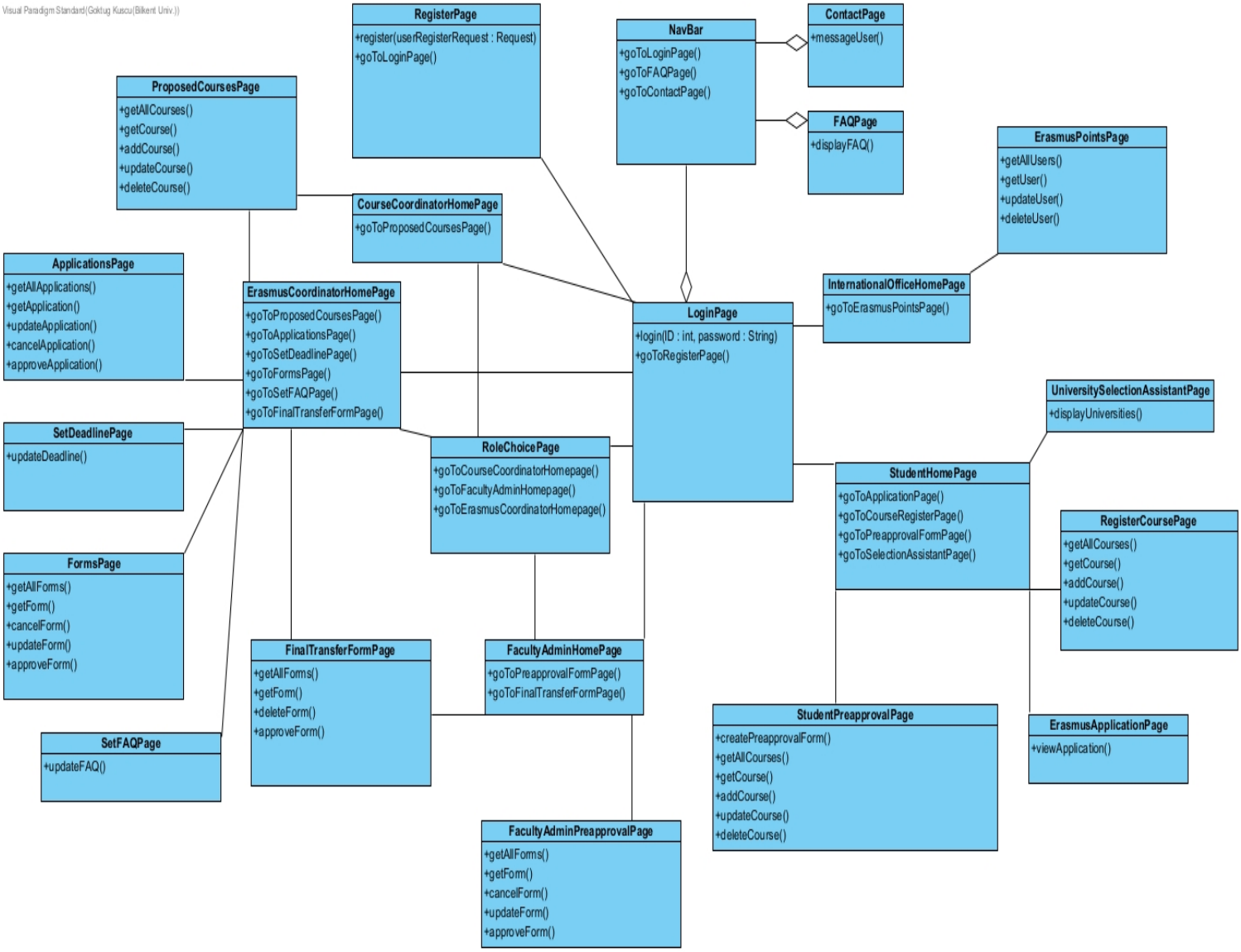
Maintainability versus Flexibility: In this application; the Model View Template architecture is used instead of the classic Model View Controller architecture. Since the views are coupled with the models in Model View Template architecture, making changes on the application is easier. However; this situation also requires the developers to be more in touch with each other's code pieces which makes it harder to resolve conflicts and failure, hence it is a disadvantage when it comes to maintainability. It is decided that flexibility is more important than maintainability since this application is a small-scale application which is being developed by a small team and the likelihood of having misinterpretations is quite high.

3.2 Final object design

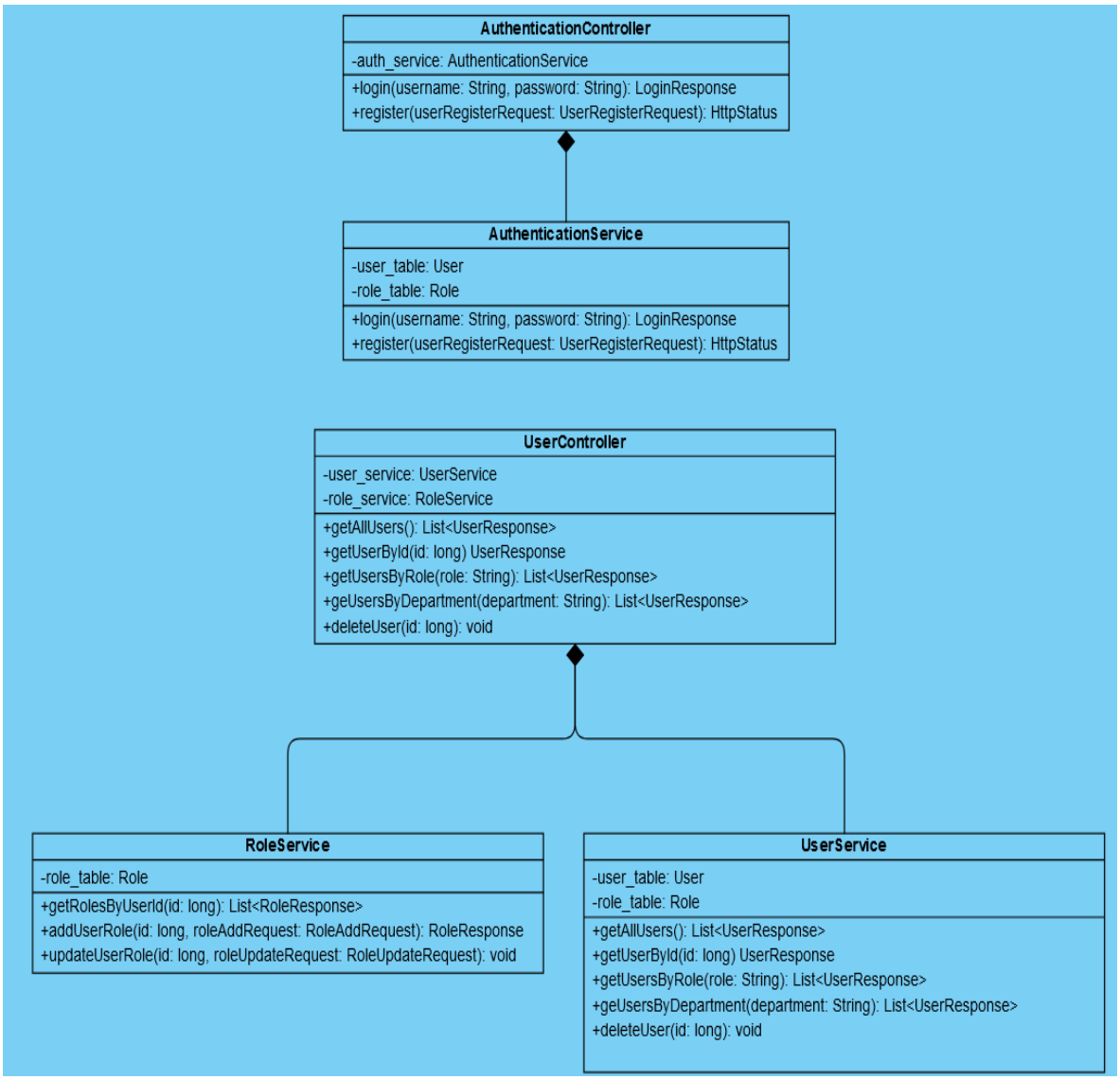


3.3. Layers

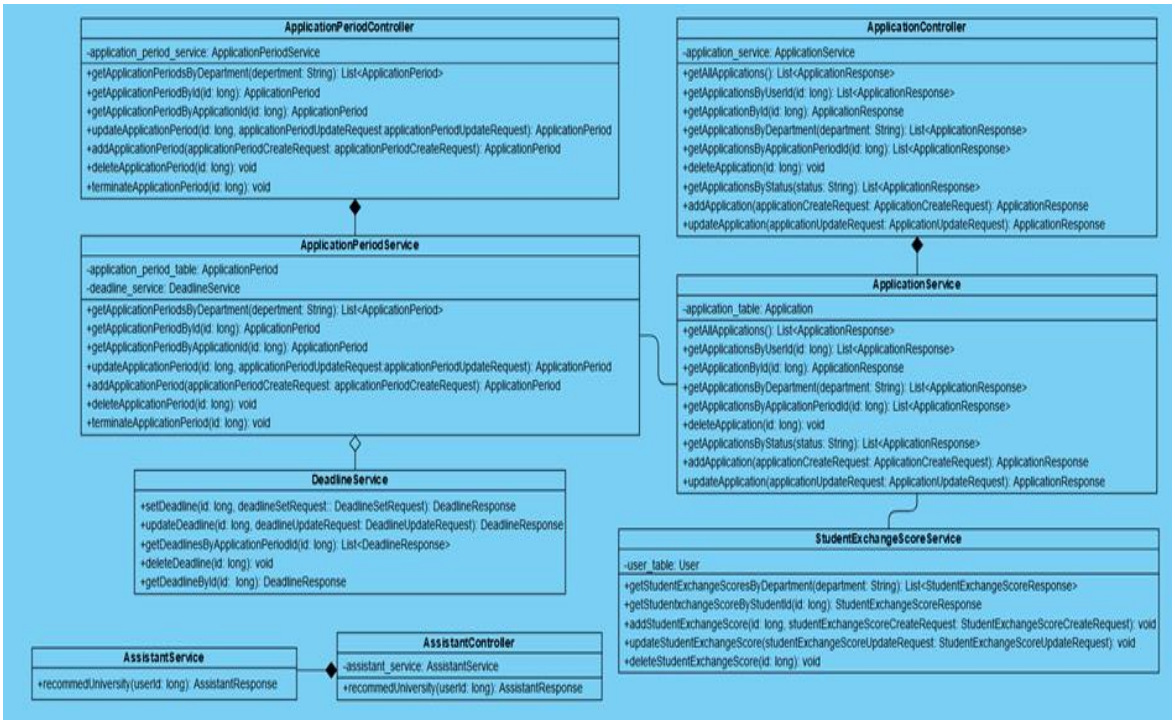
3.3.1 User Interface Management Layer



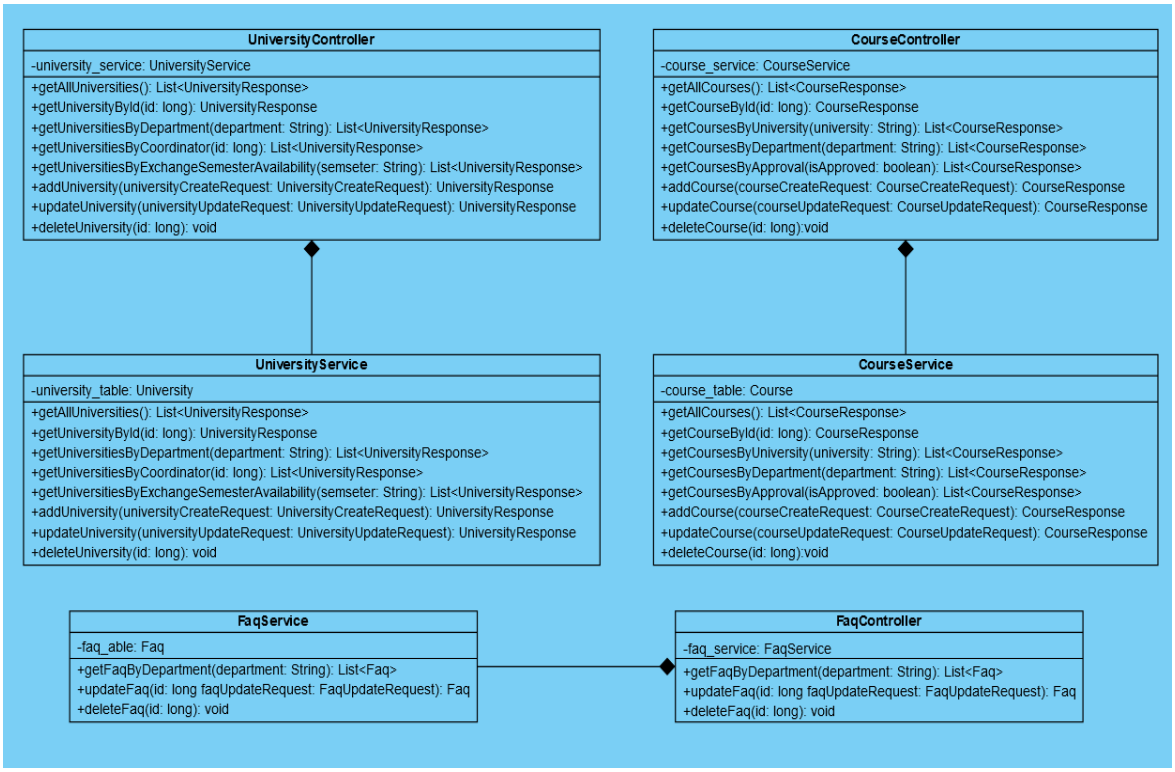
3.3.2 Web Server Layer



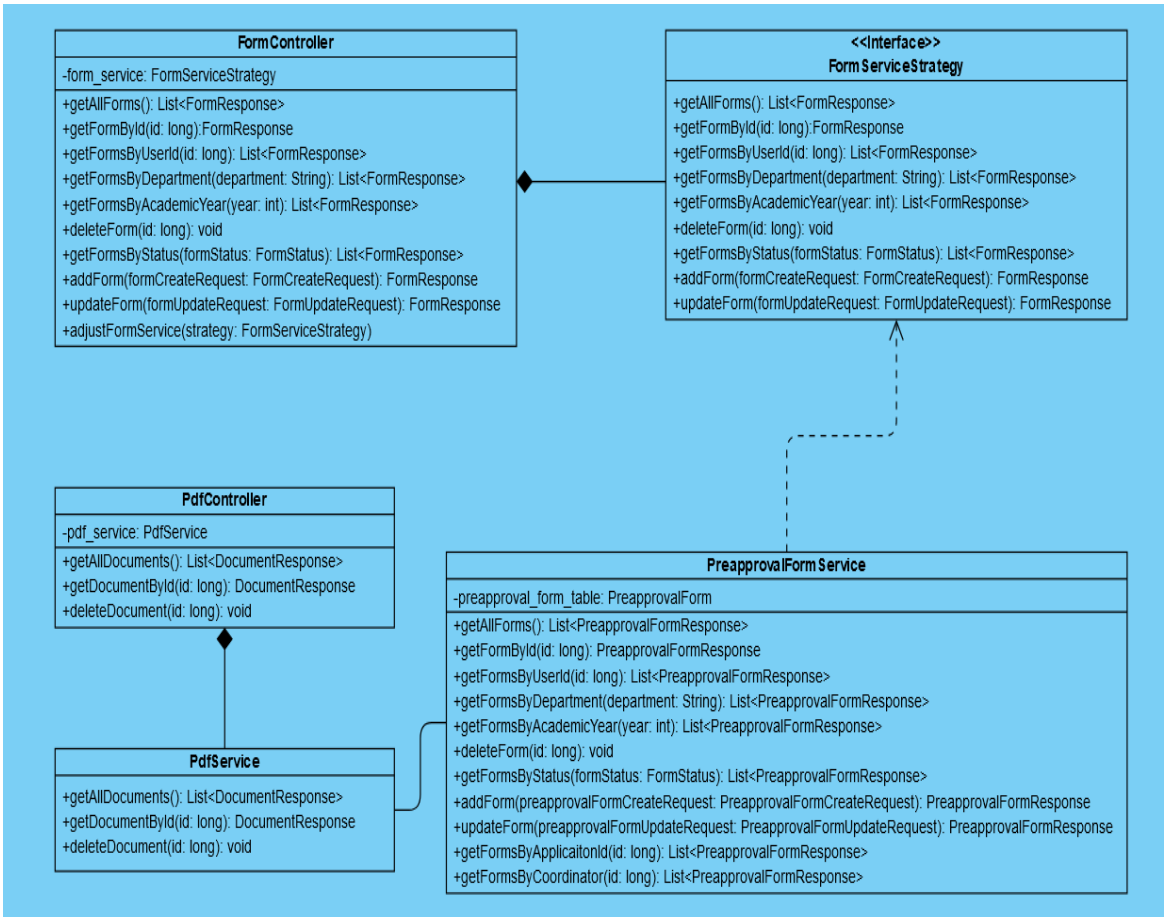
Web Server Layer's authentication and user management classes



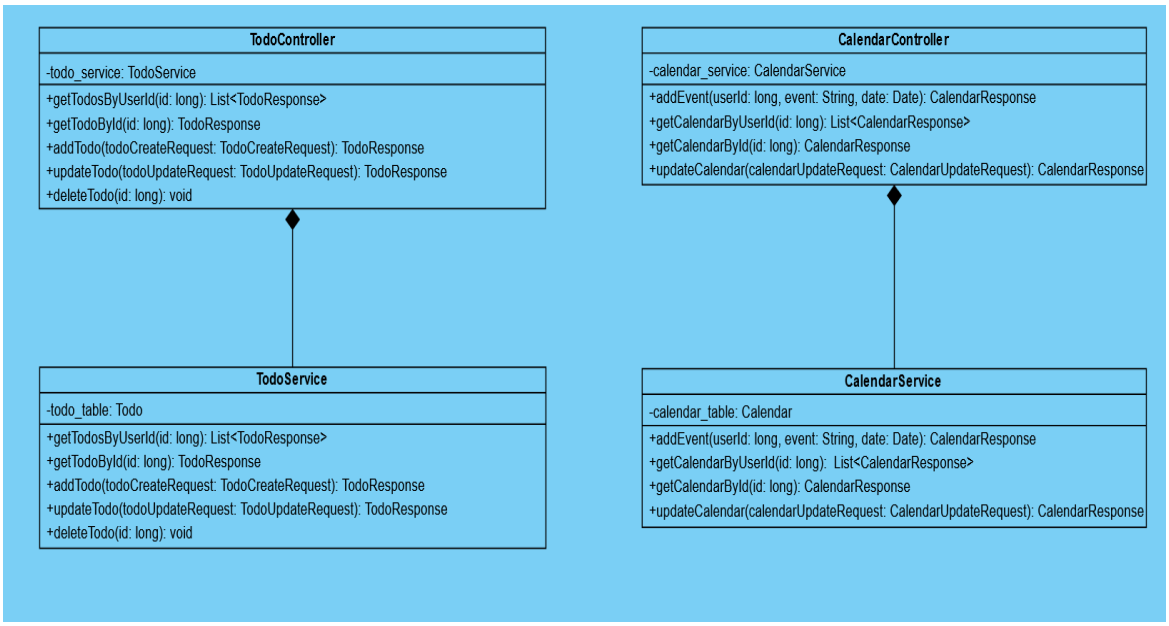
Web Server Layer's application and assistant management classes



Web Server Layer's university, course and FAQ management classes

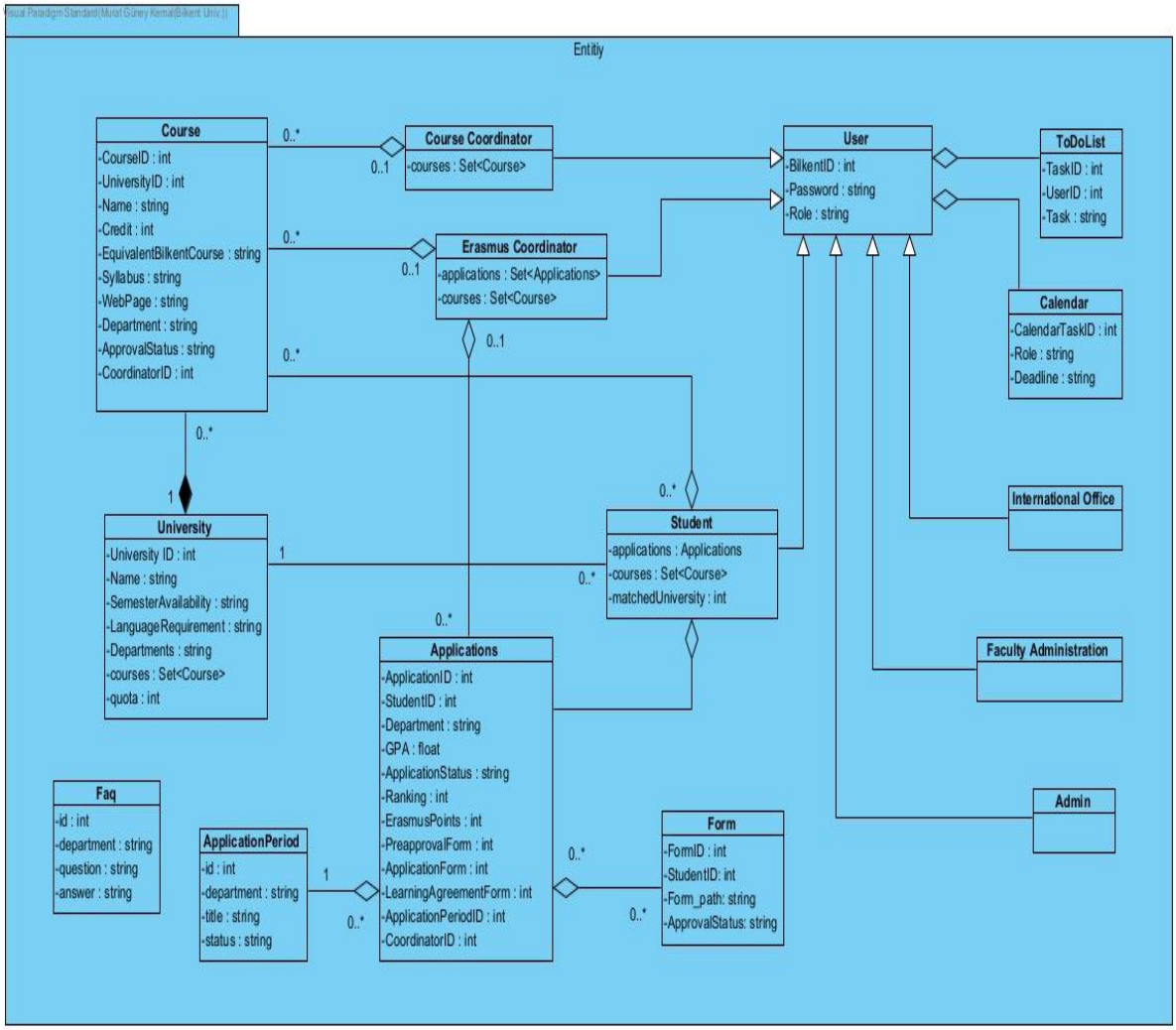


Web Server Layer's form management classes



Web Server Layer's home management classes

3.3.3. Data Management Layer



This layer represents the Data Management Layer of the application. This layer interacts with the Web Service Layer. The layer contains Database and its Entities. Database subsystems handle database communication with the help of related services. The Entities represent the tables of the database. Database communication is handled by flask and SQLAlchemy so no repository instances exist.

3.4 Design Patterns

3.4.1. Singleton Design Pattern

In the application, all of the controller objects and the service objects needed only one instance to be present at a time. Singleton design pattern allowed us to ensure that only one instance per class existed in runtime. Also the simplicity factor of the singleton design pattern made this design pattern suitable for us.

3.4.2. Strategy Design Pattern

To increase the application's feature maintainability for future changes, the strategy pattern is used. In the future the customer may require additional features that are related with another feature already present in the application. Since our application contains many forms and only one is needed at a time for a student because one form will be present in each step of the application, we have decided to use the strategy design pattern to implement Form object into Application.

3.5 External Packages

Flask-login

This package provides user session management and helps with authorization and authentication. It handles the basic tasks of logging in and logging out.

Flask-SQLAlchemy

This package is an extension of Flask which adds SQLAlchemy support. It allows the use of Object Relational Mapping (ORM) for mapping database tables to the entity objects.

Flask-Uploads

This package provides a more flexible and efficient way of handling file uploads and file downloads

Flask-Admin

This package provides a built-in admin interface for a Flask application.

Scikit-learn

This package provides built-in machine learning models which are going to be used for training the university selection assistant

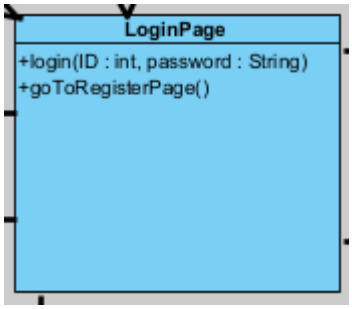
Jinja2

This package provides an extensible templating engine which makes the use of Python language in the backend replace the use of JavaScript in the frontend, hence increasing the production speed.

3.6 Class Interfaces

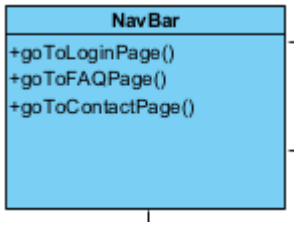
3.6.1 UI Interface Management Layer Class Interfaces

3.6.1.1 LoginPage



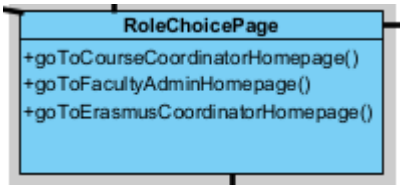
Operations:
public login(int ID, String password): On click, submits the authorization request to the website. Automatically redirects to the home page of the user's role, or if the user has multiple roles it redirects to the role choice page.
public goToRegisterPage(): On click, goes to the registration page.

3.6.1.2 NavBar



Operations:
public goToLoginPage(): On click, goes to the login page.
public goToFAQPage(): On click, goes to the FAQ page.
public goToContactPage(): On click, goes to the contacts page.

3.6.1.3 RoleChoicePage

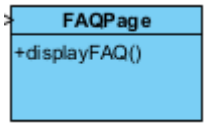


Operations:
public goToCourseCoordinatorHomepage(): On click, goes to the Course Coordinator home page.
public goToFacultyAdminHomepage(): On click, goes to the Faculty Administrator home page.
public goToErasmusCoordinatorHomepage(): On click, goes to the Erasmus Coordinator home page

3.6.1.4 ContactPage

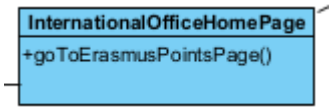
Operations:
public messageUser(): On click, allows the user to contact a user.

3.6.1.5 FAQPage



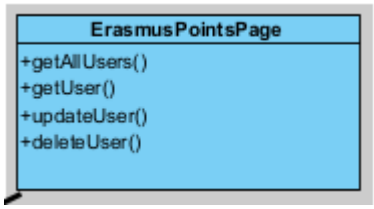
Operations:
public displayFAQ(): On click, displays the frequently asked questions.

3.6.1.6 InternationalOfficePage



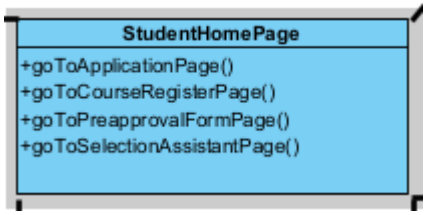
Operations:
public goToErasmusPointsPage(): On click, goes to the Erasmus points page.

3.6.1.7 ErasmusPointsPage



Operations:
public getAllUsers(): On click, displays the rankings of all students.
public getUser(): On click, displays the information of the chosen student.
public updateUser(): On click, allows a student's information to be updated.
public deleteUser(): On click, allows a student's information to be deleted.

3.6.1.8 StudentHomePage

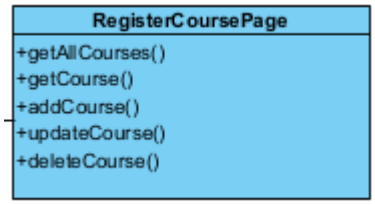


Operations:
public goToApplicationPage(): On click, goes to the application page
public goToCourseRegisterPage(): On click, goes to the course registration page
public goToPreapprovalFormPage(): On click, goes to the preapproval form page
public goToSelectionAssistantPage(): On click, goes to the selection assistant page

3.6.1.9 UniversitySelectionAssistantPage

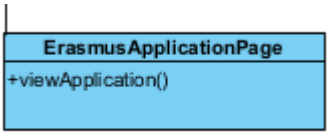
Operations:
public displayUniversities(): On click, displays where the student can expect to be placed according to previous years' placements.

3.6.1.10 RegisterCoursePage



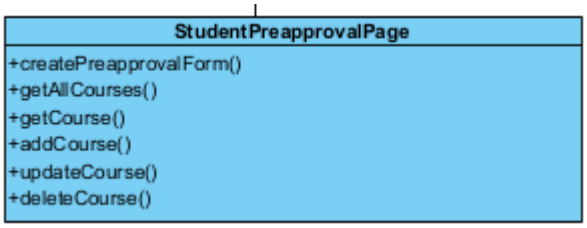
Operations:
public getAllCourses(): On click, displays all existing courses.
public getCourse(): On click, displays the chosen course.
public addCourse(): On click, allows a course to be added.
public updateCourse(): On click, allows a course to be updated.
public deleteCourse(): On click, allows a course to be deleted.

3.6.1.11 ErasmusApplicationPage



Operations:
public viewApplication(): On click, displays the user's application form.

3.6.1.12 StudentPreapprovalPage

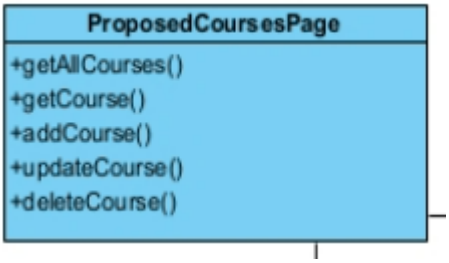


Operations:
public createPreapprovalForm(): On click, creates the application form.
public getAllCourses(): On click, displays all existing courses.
public getCourse(): On click, displays the chosen course.
public addCourse(): On click, allows a course to be added to the application.
public updateCourse(): On click, allows a course to be updated on the application.
public deleteCourse(): On click, allows a course to be deleted from the application.

3.6.1.13 RegisterPage

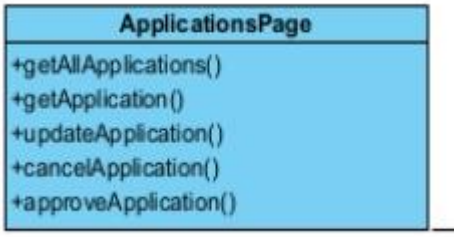
Operations:
register(userRegisterRequest request): Request stores credentials of the user, and whenever this function is used, credentials are checked from the database and on confirmation it creates a user with that credentials.
goToLoginPage() : After registering sends the user to the login page.

3.6.1.14 ProposedCoursesPage



private <CourseList> getAllCourses() : Gets all the courses that are proposed.
private course getCourse(): Gets a course with given id and university.
public void addCourse() : Adds a course to the database with given id and university, after checking whether it exists or not.
public void updateCourse(): Updates the course with given id
public void deleteCourse(): Deletes the course with given id, on success returns validation.

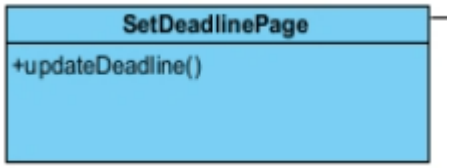
3.6.1.15 ApplicationsPage



private <ApplicationList> getAllApplication(): Returns all applications that are formed.
private application getApplication(): Returns an application with id on request.

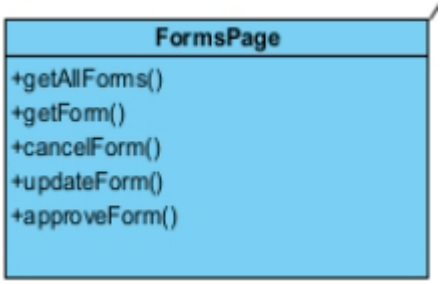
private void updateApplication(): Updated the application with the given id.
private void cancelApplicaton(): Cancels(removes) the application with given id and returns success.
private void approveApplication(): This function is used to approve or deny the application proposed.

3.6.1.16 Set DeadlinePage



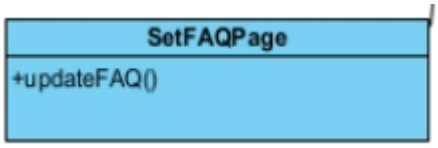
private void updateDeadline(): This function sets or updates a deadline for processes.

3.6.1.17 FormsPage



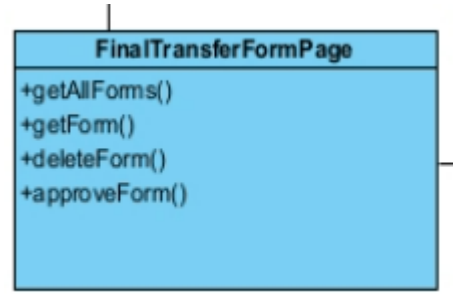
private <FormList> getAllForms(): Gets all forms created
private form getForm(): Gets a form with given id
private void cancelForm(): Cancels a form with given id and returns success.
private void updateForm(): Updates a form with given id and returns success.
private void approveForm(): This function is used to approve or deny a form with given id, returns success.

3.6.1.18 SetFAQPage



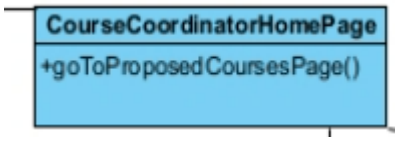
private void updateFAQ(): Adds, removes or updates question inside of FAQ.

3.6.1.19 FinalTransferFormPage



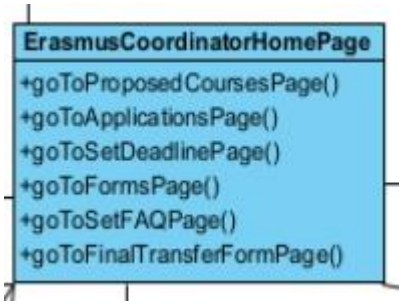
private <FormList> getAllForms(): Gets all forms created
private form getForm(): Gets a form with given id
approveForm(): This function is used to approve or deny a form with given id, returns success.

3.6.1.20 CourseCoordinatorHomePage



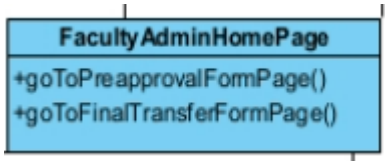
goToPropesedCoursesPage(): Directs to proposed courses page.

3.6.1.21 ErasmusCoordinatorHomePage



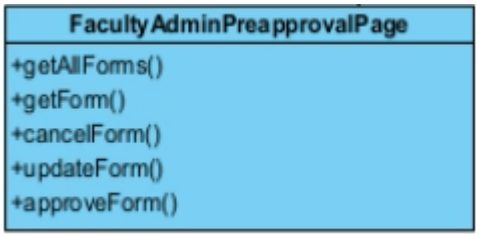
Functions inside of this page, directs the user to the page on click to the relative buttons.

3.6.1.22 FacultyAdminHomePage



Functions inside of this page, directs the user to the page on click to the relative buttons.

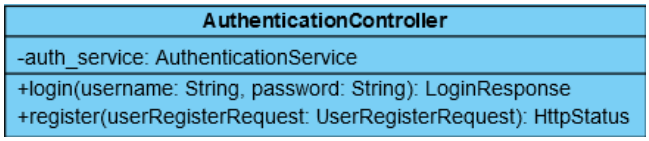
3.6.1.23 FacultyAdminPreapprovalPage



getAllForms(): Gets all forms created
getForm(): Gets a form with given id
cancelForm(): Cancels a form with given id and returns success.
updateForm(): Updates a form with given id and returns success.
approveForm(): This function is used to approve or deny a form with given id, returns success.

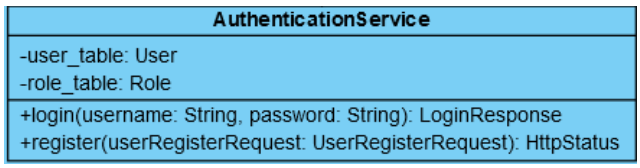
3.6.2 Web Server Layer Class Interfaces

3.6.2.1 AuthenticationController



A class to control authentication functionalities (login and register) and return the responses from the service. Registration exists only for manual user registrations to the system and is not planned to be introduced to the actual users.
Attributes:
private AuthenticationService authentication_service: Business layer object to handle the authentication functionalities.
Operations:
public LoginResponse login(String username, String password): Sends a login request to the business layer. Returns the login response sent from the authentication service.
public HTTPStatus register(UserRegisterRequest userRegisterRequest): Sends a register request to the business layer. Returns the http response (successful or unsuccessful) of the action sent from the authentication service.

3.6.2.2 AuthenticationService



A class to handle authentication functionalities (login and register) requested from the controller layer and return required responses.
Attributes:
private User user_table: Entity object that holds the user-related information including authentication information.

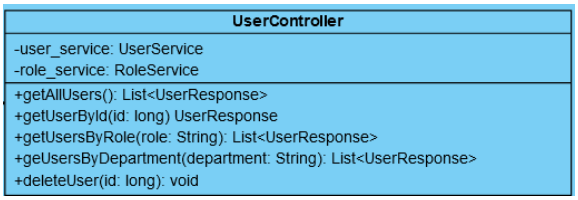
private Role role_table: Entity object that holds the role information of the users.

Operations:

public LoginResponse login(String username, String password): Checks whether a user with the given username and password credentials exist in the database. Returns the login response that includes the user object received and the role list belonging to the user.

public HTTPStatus register(UserRegisterRequest userRegisterRequest): Tries to register a new user with the given request parameter. Returns the http response (successful or unsuccessful) of the user registration action.

3.6.2.3 UserController



A class to control user-related requests and return the responses from the services.

Attributes:

private UserService user_service: Business layer object that handles the user-related data management functionalities.

private RoleService role_service: Business layer object that handles the users' role-related functionalities.

Operations:

public List<UserResponse> getAllUsers(): Sends a request to the service to get all the users registered to the system and returns a list of user responses.

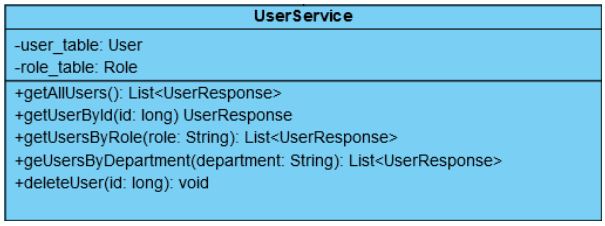
public UserResponse getUserById(): Sends a request to the service to get the user with the specific ID registered to the system and returns a user response.

public List<UserResponse> getUsersByRole(String role): Sends a request to the service to get all the users with the specific role to the system and returns a list of user responses.

public List<UserResponse> getUsersByDepartment(String department): Sends a request to the service to get all the users with the specific department to the system and returns a list of user responses.

public void deleteUser(long id): Sends a request to the service to delete the user with the specific ID.

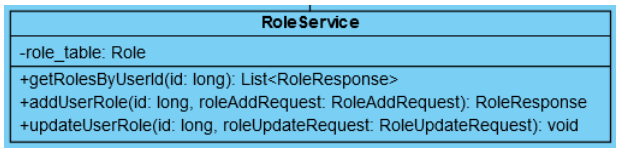
3.6.2.4 UserService



A class to handle user-related requests requested from the controller layer and return the required responses.

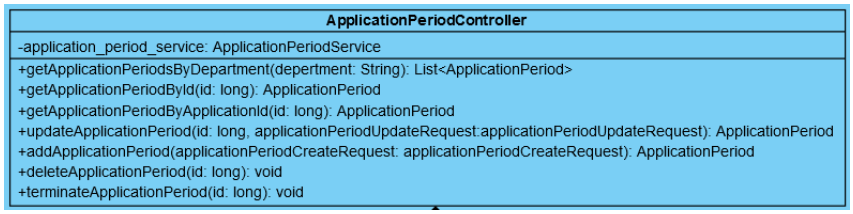
Attributes:
private User user_table: Entity object that holds the user-related information including authentication information.
private Role role_table: Entity object that holds the role information of the users.
Operations:
public List<UserResponse> getAllUsers(): Gets all the users registered to the system and returns a list of user responses.
public UserResponse getUserById(): Gets the user with the specific ID if the user exists and returns a user response.
public List<UserResponse> getUsersByRole(String role): Gets all the users with the specific role to the system and returns a list of user responses.
public List<UserResponse> getUsersByDepartment(String department): Gets all the users with the specific department to the system and returns a list of user responses.
public void deleteUser(long id): Deletes the user with the specific ID if the user exists.

3.6.2.5 RoleService



A class to handle user's role-related requests requested from the controller layer and return the required responses.
Attributes:
private Role role_table: Entity object that holds the role information of the users.
Operations:
public List<RoleResponse> getRolesByUserId(id: long): Gets all the roles related with the specific user ID and returns a list of role responses.
public RoleResponse addUserRole(long id, RoleAddRequest roleAddRequest): Adds the user with the specific id a new role with the specific request -if user exists and user does not have the requested role- and returns a role response.
public RoleResponse updateUserRole(long id, RoleUpdateRequest roleUpdateRequest): Update the user with the specific id a new role with the specific request -if user exists and user has the requested role- and returns a role response.

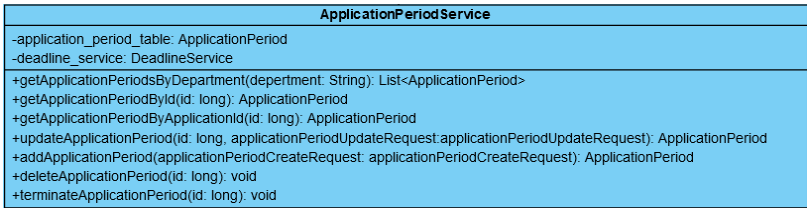
3.6.2.6 ApplicationPeriodController



A class to control application-period-related requests and return the responses from the service.

Attributes:
private ApplicationPeriodService application_period_service: Business layer object that handles the application-period-related requests.
Operations:
public List<ApplicationPeriod> getApplicationPeriodsByDepartment(String department): Sends a request to the service to get all the application periods with the specific department and returns a list of application periods.
public ApplicationPeriod getApplicationPeriodById(long id): Sends a request to the service to get the application period with the specific ID and returns an application period.
public ApplicationPeriod getApplicationPeriodByApplicationId(long id): Sends a request to the service to get the application period with the that an application belongs to with the specific ID and returns an application period.
public ApplicationPeriod addApplicationPeriod(ApplicationPeriodCreateRequest applicationPeriodCreateRequest): Sends a request to add a new application period to the system and returns an application period.
public ApplicationPeriod updateApplicationPeriod(long id, ApplicationPeriodUpdateRequest applicationPeriodUpdateRequest): Sends a request to update the application period with the specific ID according to the specified update request and returns an application period.
public void deleteApplicationPeriod(long id): Sends a request to delete the application period with the specified ID.
public void terminateApplicationPeriod(long id): Sends a request to terminate the application period with the specified ID.

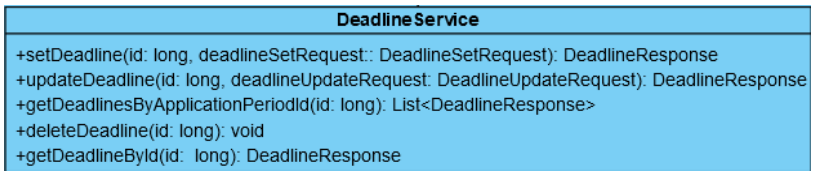
3.6.2.7 ApplicationPeriodService



A class to handle application-period-related requests and return the required responses.
Attributes:
private ApplicationPeriod application_period_table: Entity layer object that holds application periods.
private DeadlineService deadline_service: Business layer object that handles the application period deadlines.
Operations:
public List<ApplicationPeriod> getApplicationPeriodsByDepartment(String department): Gets all the application periods with the specific department and returns a list of application periods.
public ApplicationPeriod getApplicationPeriodById(long id): Gets the application period with the specific ID and returns an application period.

public ApplicationPeriod getApplicationPeriodByApplicationId(long id): Gets the application period with the that an application belongs to with the specific ID and returns an application period.
public ApplicationPeriod addApplicationPeriod(ApplicationPeriodCreateRequest applicationPeriodCreateRequest): Adds a new application period to the system and returns an application period.
public ApplicationPeriod updateApplicationPeriod(long id, ApplicationPeriodUpdateRequest applicationPeriodUpdateRequest): Updates the application period with the specific ID according to the specified update request and returns an application period.
public void deleteApplicationPeriod(long id): Deletes the application period with the specified ID.
public void terminateApplicationPeriod(long id): Updates the status of the application period with the specified ID to be terminated.

3.6.2.8 DeadlineService:

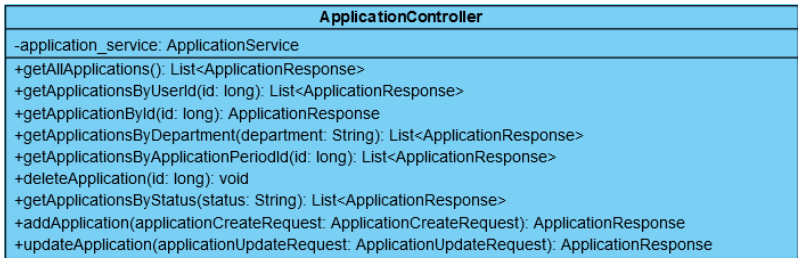


A class to handle deadline requests sent by application period service.

Operations:

public DeadlineResponse setDeadline(id: long, DeadlineSetRequest deadlineSetRequest): adds a new deadline to the application period with the specific ID according to the request and returns a deadline response.
public DeadlineResponse updateDeadline(id: long, DeadlineUpdateRequest deadlineUpdateRequest): updates the deadline of the application period with the specific ID according to the request and returns a deadline response.
public List<DeadlineResponse> getDeadlinesByApplicationPeriodId(long id): gets all the deadlines of the application period with the specific ID and returns a list of deadline responses.
public void deleteDeadline(long id): deletes the deadline with the specific ID.
public DeadlineResponse getDeadlineByID(long id): gets the deadline with specific ID and returns a deadline response.

3.6.2.9 ApplicationController



A class to control application-related requests and return the responses from the service.

Attributes:

private ApplicationService application_service: Business layer object that handles the application requests.

Operations:

public List<ApplicationResponse> getAllApplications(): Sends a request to the service to get all the applications and returns a list of application responses.

public List<ApplicationResponse> getApplicationsByUserId(long id): Sends a request to the service to get all the applications with the specific user ID and returns a list of application responses.

public ApplicationResponse getApplicationById(long id): Sends a request to the service to get the application with the specific ID and returns an application response.

public List<ApplicationResponse> getApplicationsByDepartment(String department): Sends a request to the service to get all the applications with the specific department and returns a list of application responses.

public List<ApplicationResponse> getApplicationsByApplicationPeriodId(long id): Sends a request to the service to get all the applications with the specific application period ID and returns a list of application responses.

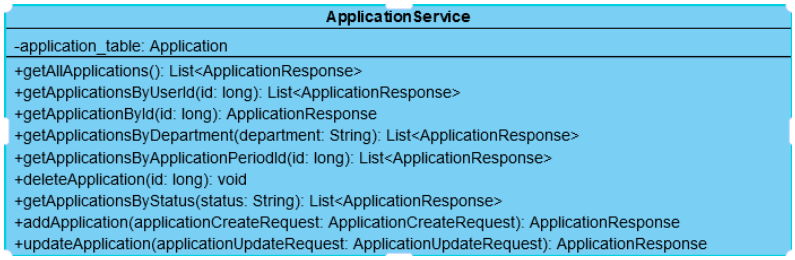
public void deleteApplicaiton(long id): Sends a request to the service to delete the application with the specific ID.

public List<ApplicationResponse> getAllApplicationsByStatus(String status): Sends a request to the service to get all the applications with the specific status and returns a list of application responses.

public ApplicationResponse addApplication (ApplicationCreateRequest applicationCreateRequest): Sends a request to the service to add the application and returns an application response.

public ApplicationResponse updateApplication(long id, ApplicationUpdateRequest applicationUpdateRequest): Sends a request to the service to update the application with the specific ID according to the specific update request and returns an application response.

3.6.2.10 ApplicationService



A class to handle application-related requests and return the required responses.

Attributes:

private Application application_table: Entity layer object that holds the applications.

Operations:

public List<ApplicationResponse> getAllApplications(): Gets all the applications and returns a list of application responses.

public List<ApplicationResponse> getApplicationsByUserId(long id): Gets all the applications with the specific user ID and returns a list of application responses.

public ApplicationResponse getApplicationById(long id): Gets the application with the specific ID and returns an application response.

public List<ApplicationResponse> getApplicationsByDepartment(String department): Gets all the applications with the specific department and returns a list of application responses.

public List<ApplicationResponse> getApplicationsByApplicationPeriodId(long id): Gets all the applications with the specific application period ID and returns a list of application responses.

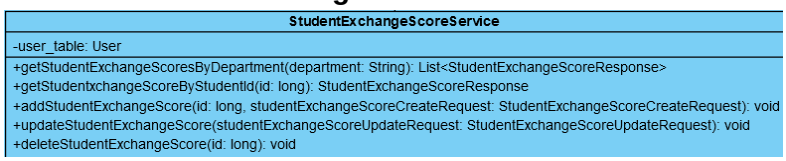
public void deleteApplicaiton(long id): Deletes the application with the specific ID.

public List<ApplicationResponse> getAllApplicationsByStatus(String status): Gets all the applications with the specific status and returns a list of application responses.

public ApplicationResponse addApplication (ApplicationCreateRequest applicationCreateRequest): Adds the application and returns an application response.

public ApplicationResponse updateApplication(long id, ApplicationUpdateRequest applicationUpdateRequest): Updates the application with the specific ID according to the specific update request and returns an application response.

3.6.2.11 StudentExchangeScoreService:



A class to handle the student exchange scores during the application process.

Attributes:

private User user_table: Entity object that holds the user-related information.

Operations:

public List<StudentExchangeScoreResponse> getStudentExchangeScoresByDepartment(String department): Gets all the Exchange

scores of the users with the specific department and returns a list of exchange score responses.
public StudentExchangeScoreResponse getStudentxchangeScoreByStudentId(long id): returns the exchange score response of the user with the specified ID.
public void addStudentExchangeScore(long id, StudentExchangeScoreCreateRequest studentExchangeScoreCreateRequest): adds the user with the specific id a new Exchange score according to the request.
public void updateStudentExchangeScore(long id, StudentExchangeScoreUpdateRequest studentExchangeScoreUpdateRequest): updates the student Exchange score of the user with the specific id according to the request if user and its score already exists.
public deleteStudentExchangeScore(id: long): deletes the student exchange score of the user with the specific ID.

3.6.2.12 AssistantController:

A class to send university-recommendation-requests to the service and responses from the service.

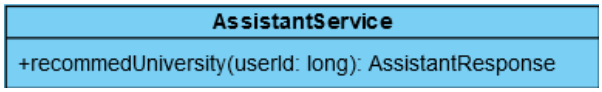
Attributes:

AssistantService assistant_service: Business layer object that handles the university-recommendation -related requests.

Operations:

public AssistantResponse recommendUniversity(id: long): Sends a request to the service and returns a response that includes a set of university recommendations according to the given application ID.

3.6.2.13 AssistantService:

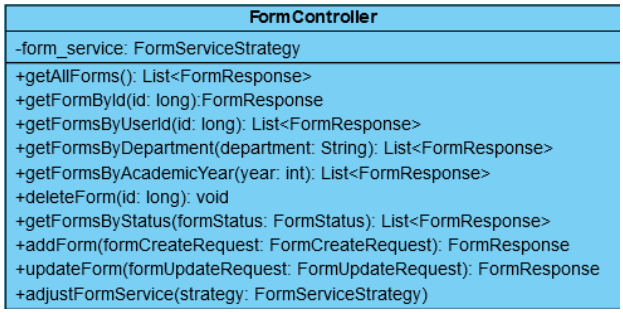


A class to handle the university recommendation feature.

Operations:

public AssistantResponse recommendUniversity(id: long): returns a response that includes a set of university recommendations according to the given application ID.

FormController:



A class tosend form related requests to the service and return the responses of the service.

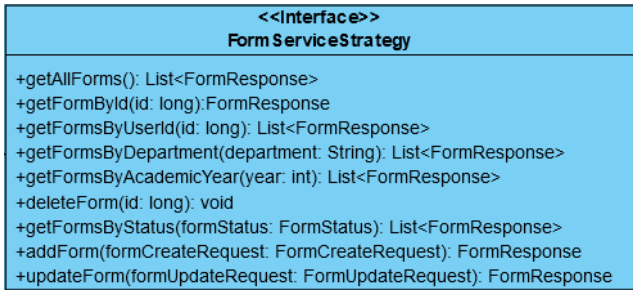
Attributes:

FormServiceStrategy form_service: This attribute will let the controller have access to its service.

Operations:

public List<FormResponse> getAllForms(): Gets all the forms that are in the system and returns them as a list.
public FormResponse getFormById(long id): Returns a form with the given form ID
public List<FormResponse> getFormsByUserId(long id): Gets all the forms that are sent by a certain user and returns them as a list.
public List<FormResponse> getFormsByDepartment(String department): Gets all the forms that are sent by the users with the given department and returns them as a list.
public List<FormResponse> getFormsByAcademicYear(int year): Gets all the forms that are sent in a given academic year and returns them as a list.
public void deleteForm(long id): Deletes the form with the given form ID.
public List<FormResponse> getFormsByStatus(FormStatus formStatus): Gets all the forms with the given status and returns them as a list.
public FormResponse addForm(FormCreateRequest formCreateRequest): Adds a form to the database with the given form create request and returns a form response.
public FormResponse updateForm(FormUpdateRequest formUpdateRequest): Updates the form with the given form update request and returns a form response.
public void adjustFormService(FormServiceStrategy strategy): Changes the form service strategy with the given strategy.

FormServiceStrategy:



An interface class to handle the form related functionalities requested from the controller and return the required responses.

Operations:

public List<FormResponse> getAllForms(): Gets all the forms that are in the system and returns them as a list.

public FormResponse getFormById(long id): Returns a form with the given form ID

public List<FormResponse> getFormsByUserId(long id): Gets all the forms that are sent by a certain user and returns them as a list.

public List<FormResponse> getFormsByDepartment(String department): Gets all the forms that are sent by the users with the given given department and returns them as a list.

public List<FormResponse> getFormsByAcademicYear(int year): Gets all the forms that are sent in a given academic year and returns them as a list.

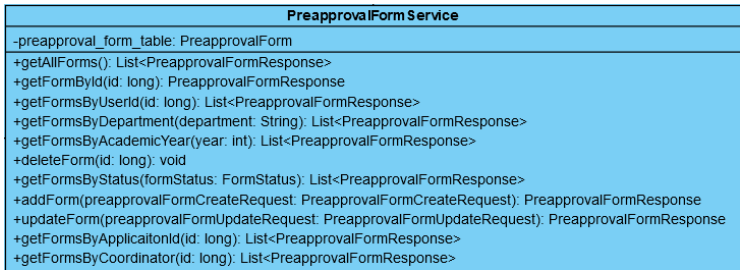
public void deleteForm(long id): Deletes the form with the given form ID.

public List<FormResponse> getFormsByStatus(FormStatus formStatus): Gets all the forms with the given status and returns them as a list.

public FormResponse addForm(FormCreateRequest formCreateRequest): Adds a form to the database with the given form create request and returns a form response.

public FormResponse updateForm(FormUpdateRequest formUpdateRequest): Updates the form with the given form update request and returns a form response.

PreapprovalFormService:



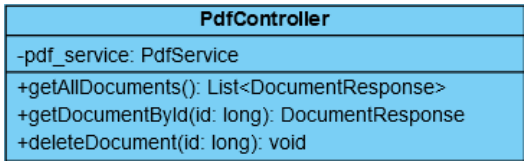
A class to handle the pre-approval form related functionalities requested from the controller and return the required responses. This class implements the FormServiceStrategy.

Attributes:

PreapprovalForm preapproval_form_table: Entity object that holds the pre-approval form related information.

Operations:
public List<PreapprovalFormResponse> getAllForms(): Gets all the pre-approval forms that are in the system and returns them as a list.
public PreapprovalFormResponse getFormById(long id): Returns a pre-approval form with the given form ID
public List<PreapprovalFormResponse> getFormsByUserId(long id): Gets all the pre-approval forms that are sent by a certain user and returns them as a list.
public List<PreapprovalFormResponse> getFormsByDepartment(String department): Gets all the pre-approval forms that are sent by the users with the given given department and returns them as a list.
public List<PreapprovalFormResponse> getFormsByAcademicYear(int year): Gets all the pre-approval forms that are sent in a given academic year and returns them as a list.
public void deleteForm(long id): Deletes the form with the given form ID.
public List<PreapprovalFormResponse> getFormsByStatus(FormStatus formStatus): Gets all the pre-approval forms with the given status and returns them as a list.
public PreapprovalFormResponse addForm(PreapprovalFormCreateRequest preapprovalFormCreateRequest): Adds a pre-approval form to the database with the given pre-approval form create request and returns a pre-approval form response.
public PreapprovalFormResponse updateForm(PreapprovalFormUpdateRequest preapprovalFormUpdateRequest): Updates the pre-approval form with the given pre-approval form update request and returns a pre-approval form response.
public List<PreapprovalFormResponse> getFormsByApplicationId(long id): Gets all the pre-approval forms that are in the same Erasmus application and returns them as a list.
public List<PreapprovalFormResponse> getFormsByCoordinator(long id): Gets all the pre-approval forms that are assigned to a given coordinator (if there are multiple coordinators all the work is allocated equally between them) and returns them as a list.

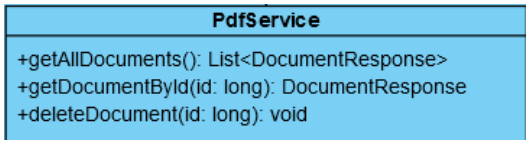
PdfController:



A class to send pdf related requests to the service and return the responses of the service.
Attributes:
PdfService pdf_service: This attribute will let the controller to have access to its service.
Operations:
public List<DocumentResponse> getAllDocuments(): Gets all the pdf documents and returns them as a list.
public DocumentResponse getDocumentById(long id): Returns a pdf document with the given document ID.

public void deleteDocument(long id): Deletes the pdf document with the given document ID.

PdfService:



A class to handle the pdf document related functionalities requested from the controller and return the required responses.

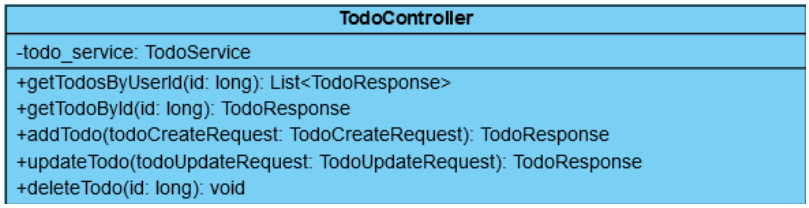
Operations:

public List<DocumentResponse> getAllDocuments(): Gets all the pdf documents and returns them as a list.

public DocumentResponse getDocumentById(long id): Returns a pdf document with the given document ID.

public void deleteDocument(long id): Deletes the pdf document with the given document ID.

TodoController:



A class to send to do list related requests to the service and return the responses of the service.

Attributes:

TodoService

todo_service: This attribute will let the controller to have access to its service.

Operations:

public List<TodoResponse> getTodosByUserId(long id): Gets all of the to do items of a given user and returns them as a list.

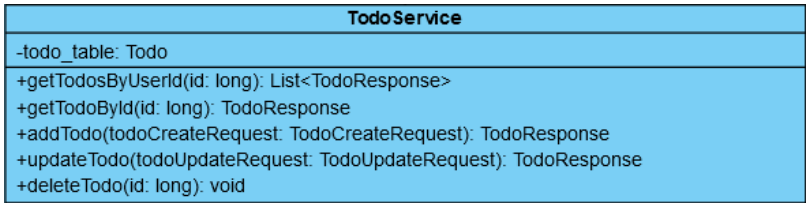
public TodoResponse getTodoById(long id): Returns a to do item with the given task ID.

public TodoResponse addTodo(TodoItemCreateRequest todoItemCreateRequest): Adds a to do item to the database with the given to do item create request and returns a to do response.

public TodoResponse updateTodo(TodoItemUpdateRequest todoItemUpdateRequest): Updates a to do item with the given to do item update request and returns a to do response.

public void deleteTodo(long id): Deletes the to do item with the given task ID.

TodoService:



A class to handle the to do item related functionalities requested from the controller and return the required responses.

Attributes:

Todo todo_table: Entity object that holds the to do item related information

Operations:

public List<TodoResponse> getTodosByUserId(long id): Gets all of the to do items of a given user and returns them as a list.

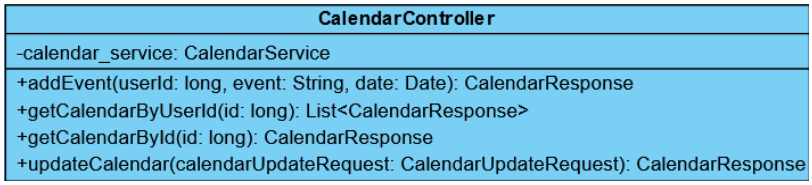
public TodoResponse getTodoById(long id): Returns a to do item with the given task ID.

public TodoResponse addTodo(TodoItemCreateRequest todoItemCreateRequest): Adds a to do item to the database with the given to do item create request and returns a to do response.

public TodoResponse updateTodo(TodoItemUpdateRequest todoItemUpdateRequest): Updates a to do item with the given to do item update request and returns a to do response.

public void deleteTodo(long id): Deletes the to do item with the given task ID.

CalendarController:



A class to send calendar related requests to the service and return the responses of the service.

Attributes:

CalendarService
calendar_service: This attribute will let the controller to have
access to its service.

Operations:

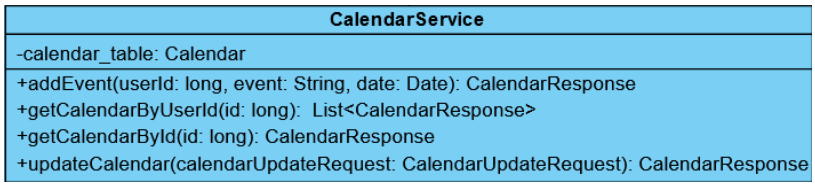
public CalendarResponse addEvent(long userId, String event, Date date): Adds an event to the calendar with the given user ID, event name and date, then returns a calendar response.

public List<CalendarResponse> getCalendarByUserId(long id): Gets the all the calendar events of the user with the given ID and returns them as a list.

public CalendarResponse getCalendarById(long id): Returns the calendar event with the given calendar task ID.

public CalendarResponse updateCalendar(CalendarUpdateRequest calendarUpdateRequest): Updates a calendar event with the given calendar update request and returns calendar response.

CalendarService:



A class to handle the calendar related functionalities requested from the controller and return the required responses.

Attributes:

Calendar calendar_table: Entity object that holds the calendar related information

Operations:

public CalendarResponse addEvent(long userId, String event, Date date): Adds an event to the calendar with the given user ID, event name and date, then returns a calendar response.

public List<CalendarResponse> getCalendarByUserId(long id): Gets the all the calendar events of the user with the given ID and returns them as a list.

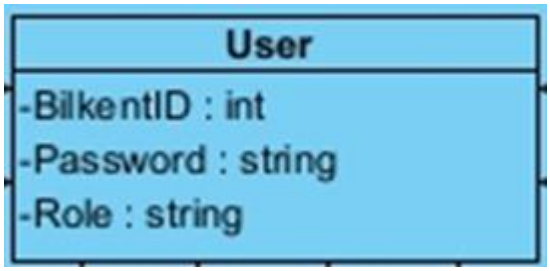
public CalendarResponse getCalendarById(long id): Returns the calendar event with the given calendar task ID.

public CalendarResponse updateCalendar(CalendarUpdateRequest calendarUpdateRequest): Updates a calendar event with the given calendar update request and returns calendar response.

3.6.3 Data Management Layer Class Interfaces

Important Note: In each entity class, an empty constructor which initializes all attributes, getter and setter methods for all the data fields exist. They are left out to improve simplicity and readability.

3.6.3.1 User



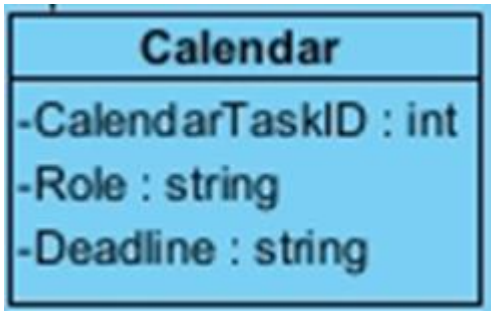
This class holds the common information of users.

Attributes:

private int BilkentID : the bilkent id of the users (primary key)

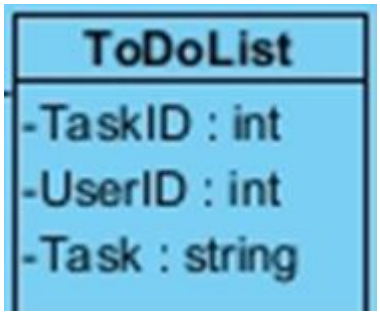
private string Password : hashed password of the users
private string Role : role or roles of the user

3.6.3.2 Calendar



This class holds the calendar events for roles. It is owned by the user with backreference.
Attributes:
private int CalendarTaskID : object's id (primary key)
private string Role : The event's related role
private string Deadline : deadline of the event

3.6.3.3 ToDoList



This class holds the tasks of users. It is owned by the user with backreference.
Attributes:
private int TaskID : object's id (primary key)
private int UserID : the bilkent id of the users
private string Task : tasks details

3.6.3.4 International Office

This class is a subclass of the user class for international office.
Attributes: Empty

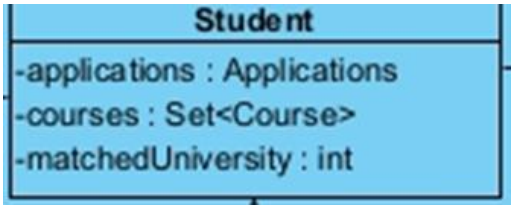
3.6.3.5 Faculty Administration

This class is a subclass of the user class for faculty administration.
Attributes: Empty

3.6.3.6 Admin

This class is a subclass of the user class for admins.
Attributes: Empty

3.6.3.7 Student

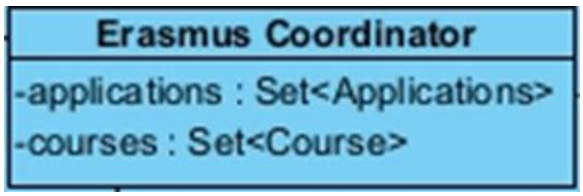


This class is a subclass of the user class for students. Has many to many relationship with Course, has one to many relationship with University and has one to one relationship with Applications. Owns Applications and Courses classes.
Attributes:
private Applications applications : student's applications
private Set<Course> courses : student's courses
private int matchedUniversity : matched university id

3.6.3.8 Course Coordinator

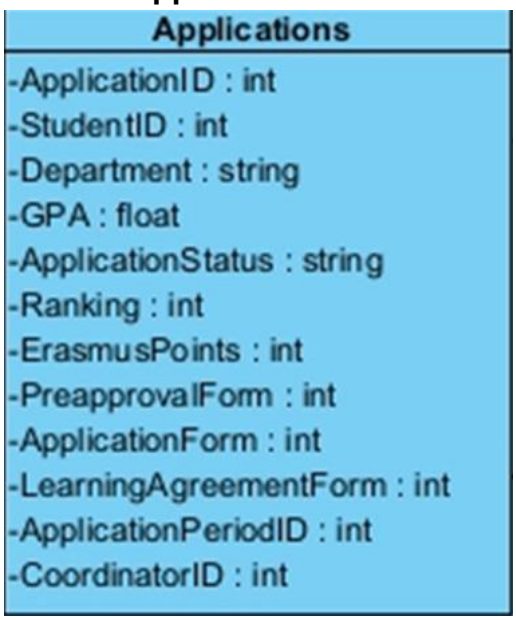
This class is a subclass of the user class for course coordinators. Has many to many relationship with Course. Owns Course class.
Attributes:
private Set<Course> courses : courses that are proposed to the course coordinator

3.6.3.9 Erasmus Coordinator



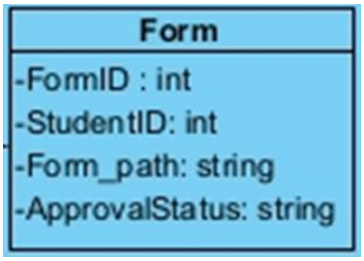
This class is a subclass of the user class for erasmus coordinators. Has many to many relationship with Course and Applications. Owns Applications and Courses classes.
Attributes:
private Set<Applications> applications : applications that are proposed to the erasmus coordinator
private Set<Course> courses : elective courses that are proposed to the erasmus coordinator

3.6.3.10 Applications



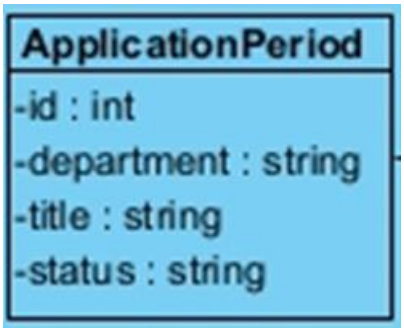
This class is for the applications of the students. Has one to one relationship with Student, has many to many relationship with Erasmus Coordinator, has one to many relationship with ApplicationPeriod, has many to many relationship with Form. Owns ApplicationPeriod and Form classes.
Attributes:
private int ApplicationID : object's id (primary key)
private int StudentID : owner student's id
private string Department : owner student's department
private float GPA : owner student's GPA
private string ApplicationStatus : current status of the application
private int Ranking : owner student's erasmus rank
private int ErasmusPoints : owner student's erasmus points
private int PreapprovalForm : owned form's id
private int ApplicationForm : owned form's id
private int LearningAgreementlForm : owned form's id
private int ApplicationPeriodID : application's period's id
private int CoordinatorD : application's reviewing coordinator's id

3.6.3.11 Form



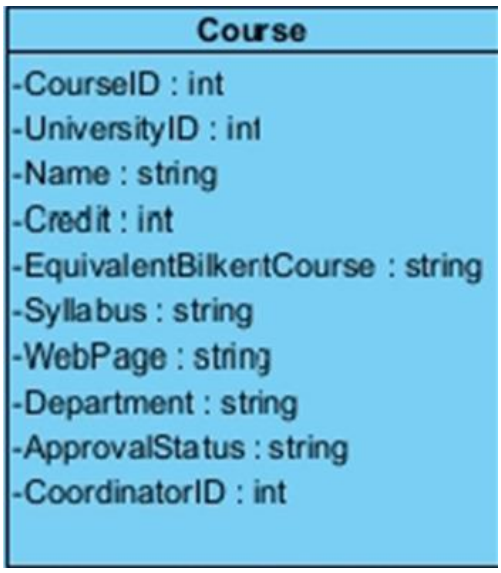
This class is for the forms of the students. Has many to many relationship with Applications.
Attributes:
private int FormID : object's id (primary key)
private int StudentID : form's owner student's id
private string Form_path : system path of the form file
private string ApprovalStatus : form's approval status

3.6.3.12 Application Period



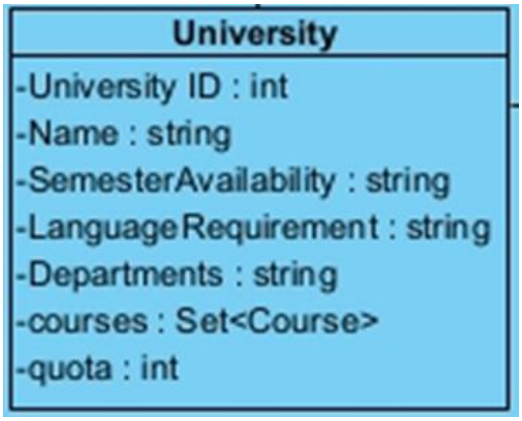
This class is for the current application period (year, semester and department). Has one to many relationship with Applications.
Attributes:
private int id : object's id (primary key)
private string department : application period's department
private string title : application period's department
private string status : application period's status

3.6.3.13 Course



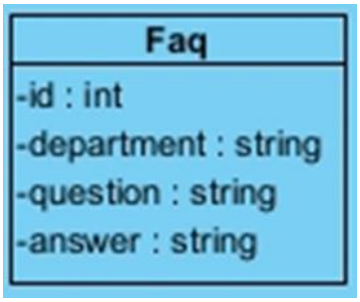
This class is for the partner university courses. Has many to many relationship with Student, has one to many relationship with Erasmus Coordinator, has one to many relationship with Course Coordinator, has one to many relationship with University.
Attributes:
private in CourseID : object's id (primary key)
private int UniversityID : course's owner university's id
private string Name : course's name
private int Credit : course's credit
private string EquivalentBilkentCourse : course's equivalent Bilkent course
private string Syllabus : course's syllabus' system path
private string WebPage : course's webpage
private string Department : course's department
private int CoordinatorID : course's bilkent equivalent coordinator's id

3.6.3.14 University



This class is for the partner universities. Has one to many relationship with Student, has one to many relationship with Course. Contains Course class.
Attributes:
private int UniversityID : course's owner university's id
private string Name : course's name
private string SemesterAvailability : available semesters of the university
private string LanguageRequirement : language requirement of the university
private string Departments : university's student accepting departments
private Set<Course> courses : university's previously accepted courses
private int quota : university's remaining quota

3.6.3.15 Faq



This class is for faq items.
Attributes:
private int id : object's id (primary key)
private string department : department of the faq item
private string question : question of the faq item
private string answer : answer of the faq item

4. Improvement Summary

1.2.5 Security: Section is updated based on the feedback.
2.1 Subsystem Decomposition is updated based on feedback.
2.2 Deployment diagram is added
2.4 Access matrix is added.
3.3.2 Web Service layer: Classes are updated and typos are fixed.
3.3.3. Data Management Layer: Entity diagram and the explanation are updated. Entity relation diagram is deleted to reduce redundancy.
3.4 Design Patterns: New design patterns are added. (Template, Singleton, Strategy)
3.6 Class Interfaces: This section is added to further explain classes.