Bilkent University

Department of Computer Engineering

CS319 Term Project

Project short-name: BEAM (Bilkent Erasmus Application Manager)

Project Design Report

Group 1F: Yağız Can Aslan, Can Ersoy, Selim Can Güler, İlkim Elif Kervan, Kemal Kubilay Yılmaz

Instructor: Eray Tüzün

Teaching Assistant(s): Emre Sülün, İdil Hanhan, Mert Kara, Muhammad Umair Ahmed, Metehan Saçakçı

Design Report Iteration 2

December 11, 2022

1. Introduction

1.1. Purpose of the System

BEAM is a web application that aims to automate and organize the application process of the Erasmus Program. Currently, almost every operation about the application to the Erasmus Program is done through emails, which slows down the process. Also, Bilkent Coordinators have to deal with a lot of paperwork that could be easily done through a digital system. BEAM offers to ease this process by allowing the actors to complete and track their tasks in an organized manner through a single application. Moreover, outgoing students can find relevant information about the universities they would like to apply to; courses they would like to take during their Erasmus Program; the experienced students provide detailed information and comment about both the universities and the courses.

1.2. Design Goals

1.2.1. Automation

As aforementioned, one of the top priority goals of BEAM is to automate the application process of the Erasmus Program. Unnecessary back-and-forth mailing between the actors is to be removed from the process to reduce the workload of both the Bilkent Coordinators and the students. The system will complete tasks that can be done automatically, such as generating pre-filled forms, including pre-approval forms and learning agreement forms. The system will automatically track the approval state of these forms.

1.2.2.Usability

BEAM's user interface (UI) is designed in such a way that the user can reach all of the authorized functionalities of the system easily. As the application process is long and complex, the user should be able to navigate through the system without being confused. Considering these factors, any functionality will be accessible with a maximum of 3 clicks by the user. To decrease the cluttering of functionalities, the main functionalities will be grouped on the sidebar with only the actions specific to the user type displayed.

2. High Level Software Architecture

2.1. Subsystem decomposition

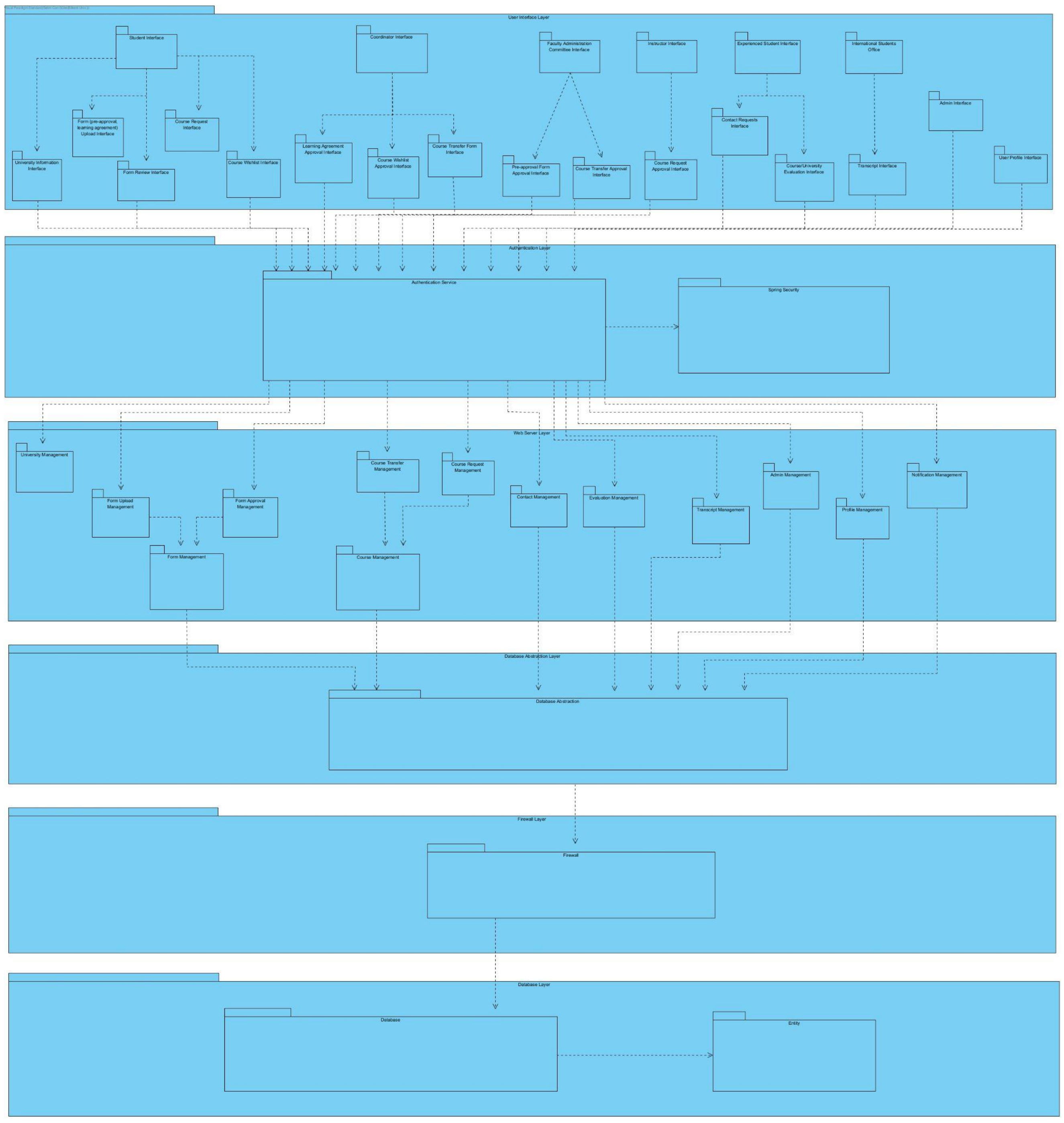

User Interface Layer
Student Interface
Coordinator Interface
Faculty Administration Committee Interface
Instructor Interface
Experienced Student Interface
International Students Office
Admin Interface
Form (pre-approval, learning agreement) Upload Interface
Course Request Interface
Learning Agreement Approval Interface
Course Wishlist Approval Interface
Course Transfer Form Interface
Contact Requests Interface
University Information Interface
Form Review Interface
Course Wishlist Interface
Pre-approval Form Approval Interface
Course Transfer Approval Interface
Course Request Approval Interface
Course/University Evaluation Interface
Transcript Interface
User Profile Interface
Authentication Layer
Authentication Service
Spring Security
Web Server Layer
University Management
Course Transfer Management
Course Request Management
Form Upload Management
Form Approval Management
Contact Management
Evaluation Management
Admin Management
Notification Management
Transcript Management
Profile Management
Form Management
Course Management
Database Abstraction Layer
Database Abstraction
Firewall Layer
Firewall
Database Layer
Database
Entity

Figure 1. Subsystem Decomposition Diagram

Our architecture has 6-layers: User Interface Layer, Authentication Layer, Web Server Layer, Database Abstraction Layer, Firewall Layer, and Database Layer. These layers are created by taking the separation of concerns into consideration. These layers will allow a more scalable coding experience since each layer will be independent. The Authentication Layer and Firewall Layer will increase the security of the system. Whereas Database Abstraction Layer is there as part of the Facade design pattern.

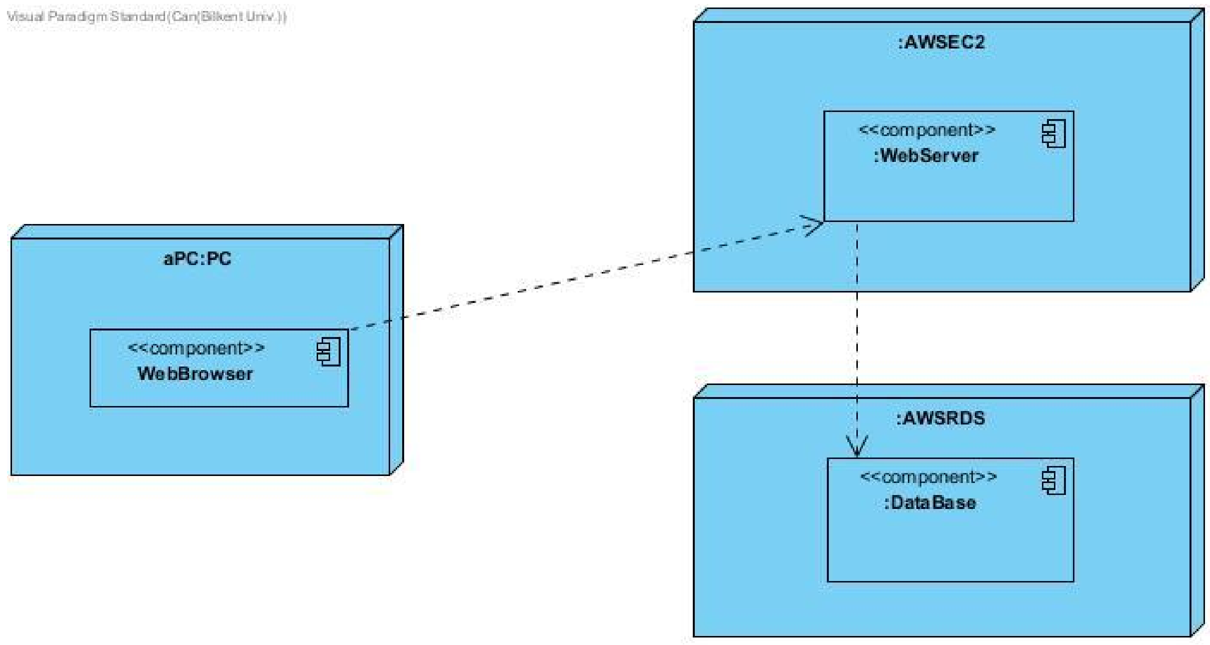
Higher resolution of this diagram can be found here:
http://elif.kervan.ug.bilkent.edu.tr/cs319_design_report/Subsystem_Decomposition.jpg

2.2. Hardware/Software Mapping

We will use React with TypeScript for Beamfrontend. TypeScript supports transpiling to older versions of JavaScript, which provides backwards-compatibility. We also indirectly use HTML5 and CSS3, which are supported by all modern browsers. Therefore, Beamdoes not demand high specifications in terms of hardware, the only requirement is a stable version of a modern browser.

Our backend will use Java, Spring Boot, and PostgreSQL. The backend and the database will be deployed to AWS servers, and static files such as forms will be stored on Amazon S3. The number of computer science students who applied for Erasmus in 2021 was around 50. Taking this number as an estimate, we expect a maximum of 1000 Erasmus applicants. This number decreases when we consider the number of concurrent users. Because of these numbers, we think the free tier server offered by AWS's EC2 service with the specifications of 1 GB memory and a 3.1 GHz Intel Xeon processor will be enough [1]. The database will be kept on a separate machine using the AWS RDS service [2]. These configurations could be quickly upgraded in case these specifications are not enough.

As a side note, external libraries for both the backend and the frontend will be utilized. Although highly unlikely, this might cause compatibility issues on certain but limited client-side hardware.



In the above diagram, the WebBrowser component contains the User Interface Layer; the WebServer component contains the Authentication Layer, Web Server Layer, Database

Abstraction Layer, and Firewall Layer; and the DataBase component contains the Database Layer.

Figure 2. Deployment Diagram

2.3. Persistent Data Management

BEAM is highly dependent on complex data such as courses from both Bilkent and other universities, different types of users, host universities, and various files and images. Therefore it is crucial to have a secure and efficient database system to manage these data. We chose a relational database instead of NoSQL since it is more secure, reliable, and familiar [3]. Afterwards, we agreed to use PostgreSQL because it supports objects and many other data types that other relational database management systems do not support [3]. Furthermore, PostgreSQL deals with concurrency better than other systems, such as MySQL, which can enhance system performance if the number of users simultaneously using BEAM increases [3]. We will use PostgreSQL for the user, course, and university-related data. Moreover, other data, such as images and documents, will be stored in the remote file system and will be accessed via file paths stored in the database.

2.4. Access Control and Security

Our Erasmus Application Manager, Beam, prioritizes security and establishes an access control system. The access control system is managed by attaining roles for users. Each of the users has certain roles assigned to them. These roles are outgoing student, incoming student, experienced student, coordinator, instructor, international students office (OISEP) member, and faculty administration committee (FAC) member.

On the client side, we use protected routes to restrict the access of different user types to certain user interfaces. This means that users can only see the user interface relevant to them and nothing more. We handle the protected routes by using the user type information sent from the backend once the user is logged in and restrict each user with predefined routes specific to their user type. This ensures that nothing wrong or irrelevant is displayed on the client side. For example, a student is not able to see the course transfer approval page since that page is protected and is only visible to a FAC member,

On the server side, we implement a system that assigns a user's role once they are registered to the system. Later, when a user tries to log in, the user type information is accessed from the database and sent to the client side. Moreover, each user can only make specific requests to the server side. We ensure this by restricting certain functionality to certain user types. For example, a student is not able to approve a pre-approval form. And a coordinator is not able to create a pre-approval form.

In terms of security, we encrypt sensitive user information before storing them in the database. Apart from this, we implement a six-layer architecture that applies restricted access to our database and backend. The six-layer architecture enforces that only our backend server can access or modify the backend. This is done via a firewall that validates requests to the backend. This makes sure that the database remains secure.

Access Control Matrix

	Outgoing Student	Incoming Student	Coordinator	Instructor	FAC Member	OISEP Staff	Admin
Login	X	X	X	X	X	X	X
Renew password	X	X	X	X	X	X	X
Create course transfer form			X				
See courses in Bilkent	X	X	X				
Access any student transcript			X			X	X
See student course wishlist			X				
Approve/reject student course wishlist			X				
Give feedback on student course wishlist			X				
See universities list	X	X	X	X	X	X	X
See university details	X	X	X	X	X	X	X
Create course wishlist	X						
Make course request	X						

See previous course requests	**X**						
Download pre-approval form	**X**		**X**				
Upload pre-approval form	**X**						
Generate Pre-approval form	**X**						
See pre-approval form	**X**		**X**				
Approve/Reject pre-approval form			**X**				
Approve/Reject course transfer					**X**		
View profile	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Edit profile	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Register user							**X**
Register users in bulk							**X**
Add/remove/ edit courses							**X**
Add/remove/ edit universities							**X**

2.5. Global Control Flow

We use event-driven control for our global control flow. The actions of the users determine the flow of the application. The users will interact with boundary objects, and those interactions will dispatch certain events on the backend. Then, the user will get a response.

We use a decentralized design in terms of the structure of our control mechanism. The control on the backend side is distributed among multiple objects. Which makes it easier to create a scalable backend.

2.6. Boundary Conditions

Here is an overview of the boundary conditions of our system. Ideally, we are aiming to have minimal failures. And for the system failures, we would like to have a recovery mechanism that will bring us back to the stable state.

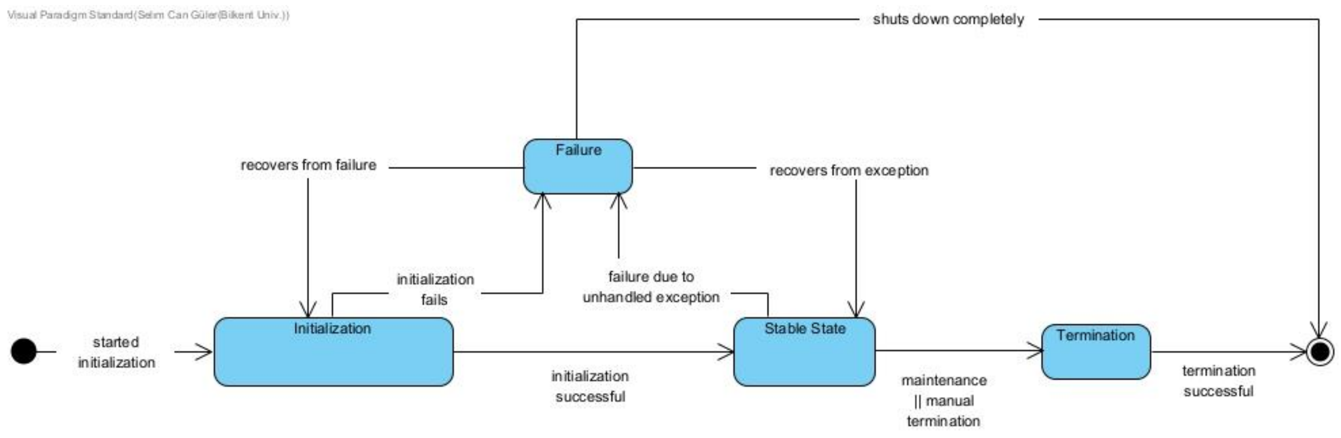


Figure 3. Boundary Conditions State Diagram

2.6.1. Initialization

Since BEAM is a web application, it is only necessary to log in to the website with an existing account in order to initialize the application. Once the user logs in, the relevant data is fetched from the database to initialize the system. Users who do not have accounts may have access to the application; however, they will have limited access, such as only seeing university information pages. When a user logs in to the application, the related page options appear on the sidebar. When a page is clicked, the information that the page contains, such as added course list, student list, or student pre-approval forms, is fetched from the database. Moreover, in order to stand the server up, the maintainer needs to initialize the server on AWS. Also, since one of the crucial parts of the application handling various information and uploaded files of different users, AWS S3 credentials should be provided and AWS S3 setup should be checked. Lastly, to ensure the initialization, the database should be initialized and correct credentials should be passed to Beam. If necessary, the IP of the server should be provided to the database so that the database will only respond to our backend. In addition to those, when a user signs in an access and refresh token is created for them internally. These tokens are later used for checking information integrity.

2.6.2. Termination

BEAM can be terminated in four possible ways. Firstly, if a user logs out of the application, the application terminates with the last saved data of the user stored in the database.

Secondly, an admin user may terminate the program for different purposes, such as maintenance or if there is a security concern. In that case, the last submitted version of the users' data is saved to the database, and the application terminates. Thirdly, a failure due to an unhandled exception may occur in the system, forcing the application to terminate. However, these types of situations are mostly handled in our backend, which either makes the system exit gracefully or shows helpful errors to the user.

Moreover, when a user’s access token expires, a request is sent to the authentication service of the application. The authentication service refreshes the access token of the user if the refresh token has not expired yet. Otherwise, when the access token expires, the user will be automatically signed off.

2.6.3. Failure

BEAM will run on AWS EC2 servers, and there is no spare web server to run BEAM on. So, the application may fail if a problem occurs in AWS. In that case, the program is restarted with the last saved data in the database. Moreover, since BEAM is a web application, the internet connection loss of the user may also lead to failure. In that case, the user needs to log in again, and the application starts with the last saved data for the user from the database.

3. Low Level Design

3.1. Object Design Trade-offs

Functionality v. Usability: The application process is complex and long. Therefore, the users should not be confused while trying to complete a task or they should not get lost through the app while trying to get some information about their application. Therefore BEAM should be as easy to use as possible.

Security v. Usability: Users of BEAM will enter the application by their email and password and no 2 Factor Authentication will be used. This reduces the application's security, yet it increases BEAM’s usability.

Rapid Development v. Automation: One of the primary goals of BEAM is to decrease the users' workload and reduce the unnecessary back-and-forth mailing between the actors. Therefore, we aim to have an automated system that will ease the application process. Hence, we will make the application as automated as possible, which is a tradeoff for rapid development as it increases the time that is needed to develop the application.

3.2. Final Object Design

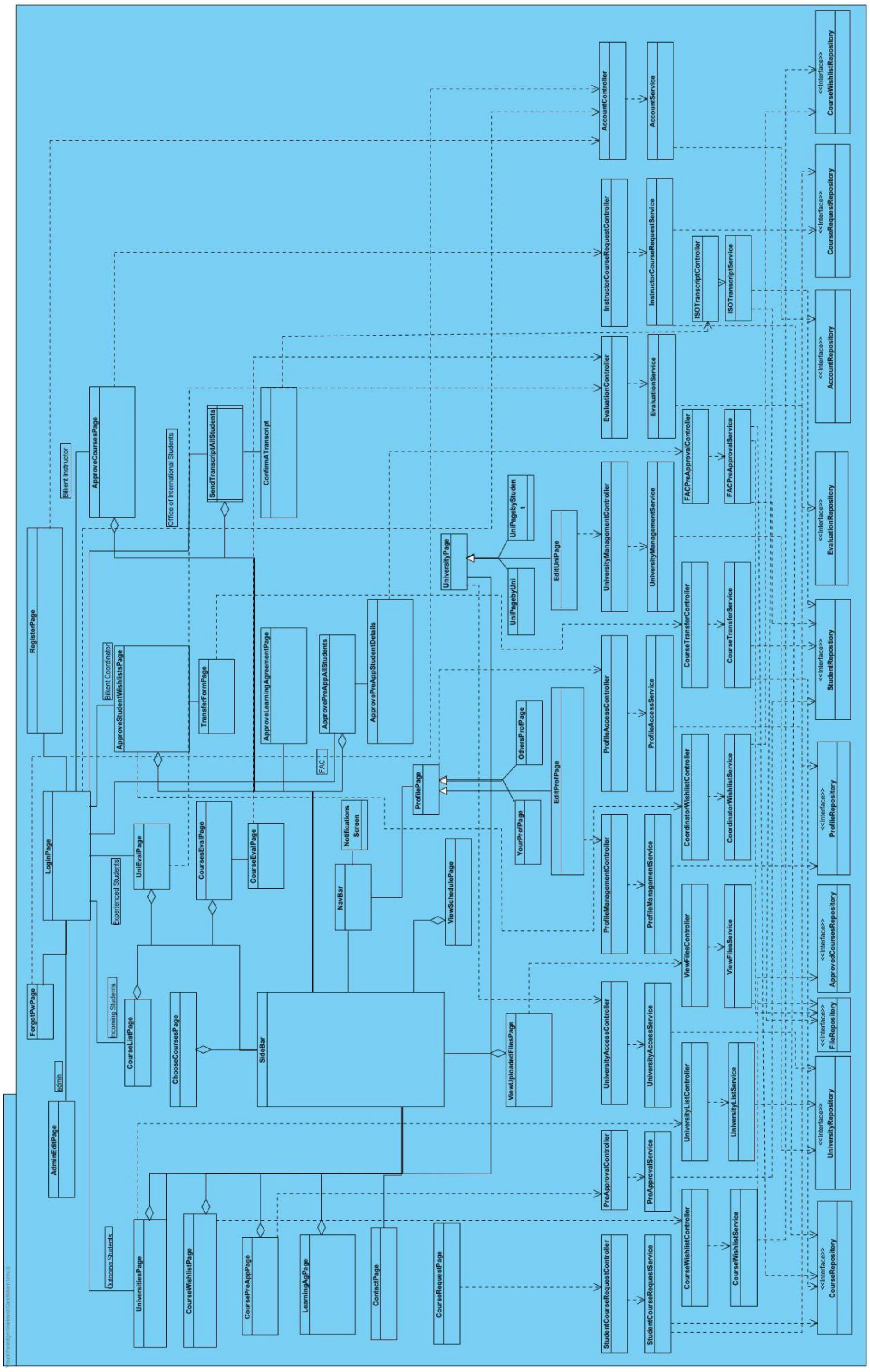


Figure 4. Final Object Design Diagram

Higher resolution of this diagram can be found [here](http://elif.kervan.ug.bilkent.edu.tr/cs319_design_report/FinalObject.jpg):
http://elif.kervan.ug.bilkent.edu.tr/cs319_design_report/FinalObject.jpg

3.3. Layers

3.3.1. User Interface Management Layer

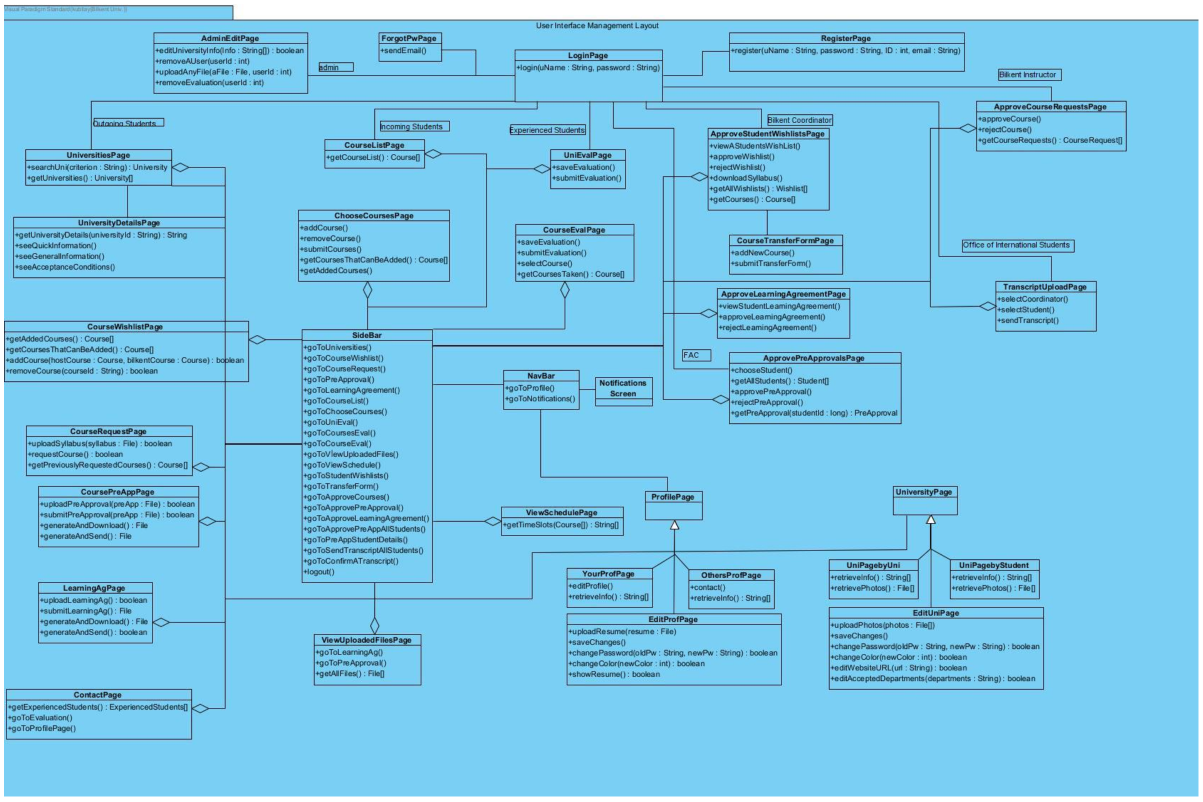

User Interface Management Layout
AdminEditPage
+editUniversityInfo(Info : String[]) : boolean
+removeAUser(userId : int)
+uploadAnyFile(aFile : File, userId : int)
+removeEvaluation(userId : int)
ForgotPwPage
+sendEmail()
admin
LoginPage
+login(uName : String, password : String)
RegisterPage
+register(uName : String, password : String, ID : int, email : String)
Bilkent Instructor
Outgoing Students
Incoming Students
Experienced Students
Bilkent Coordinator
ApproveCourseRequestsPage
+approveCourse()
+rejectCourse()
+getCourseRequests() : CourseRequest[]
UniversitiesPage
+searchUni(criterion : String) : University
+getUniversities() : University[]
CourseListPage
+getCourseList() : Course[]
UniEvalPage
+saveEvaluation()
+submitEvaluation()
ApproveStudentWishlistsPage
+viewAStudentsWishList()
+approveWishlist()
+rejectWishlist()
+downloadSyllabus()
+getAllWishlists() : Wishlist[]
+getCourses() : Course[]
UniversityDetailsPage
+getUniversityDetails(universityId : String) : String
+seeQuickInformation()
+seeGeneralInformation()
+seeAcceptanceConditions()
ChooseCoursesPage
+addCourse()
+removeCourse()
+submitCourses()
+getCoursesThatCanBeAdded() : Course[]
+getAddedCourses()
CourseEvalPage
+saveEvaluation()
+submitEvaluation()
+selectCourse()
+getCoursesTaken() : Course[]
CourseTransferFormPage
+addNewCourse()
+submitTransferForm()
Office of International Students
ApproveLearningAgreementPage
+viewStudentLearningAgreement()
+approveLearningAgreement()
+rejectLearningAgreement()
TranscriptUploadPage
+selectCoordinator()
+selectStudent()
+sendTranscript()
CourseWishlistPage
+getAddedCourses() : Course[]
+getCoursesThatCanBeAdded() : Course[]
+addCourse(hostCourse : Course, bilkentCourse : Course) : boolean
+removeCourse(courseId : String) : boolean
SideBar
+goToUniversities()
+goToCourseWishlist()
+goToCourseRequest()
+goToPreApproval()
+goToLearningAgreement()
+goToCourseList()
+goToChooseCourses()
+goToUniEval()
+goToCoursesEval()
+goToCourseEval()
+goToViewUploadedFiles()
+goToViewSchedule()
+goToStudentWishlists()
+goToTransferForm()
+goToApproveCourses()
+goToApprovePreApproval()
+goToApproveLearningAgreement()
+goToApprovePreAppAllStudents()
+goToPreAppStudentDetails()
+goToSendTranscriptAllStudents()
+goToConfirmATranscript()
+logout()
FAC
NavBar
+goToProfile()
+goToNotifications()
Notifications Screen
ApprovePreApprovalsPage
+chooseStudent()
+getAllStudents() : Student[]
+approvePreApproval()
+rejectPreApproval()
+getPreApproval(studentId : long) : PreApproval
CourseRequestPage
+uploadSyllabus(syllabus : File) : boolean
+requestCourse() : boolean
+getPreviouslyRequestedCourses() : Course[]
CoursePreAppPage
+uploadPreApproval(preApp : File) : boolean
+submitPreApproval(preApp : File) : boolean
+generateAndDownload() : File
+generateAndSend() : File
ViewSchedulePage
+getTimeSlots(Course[]) : String[]
ProfilePage
UniversityPage
YourProfPage
+editProfile()
+retrieveInfo() : String[]
OthersProfPage
+contact()
+retrieveInfo() : String[]
UniPagebyUni
+retrieveInfo() : String[]
+retrievePhotos() : File[]
UniPagebyStudent
+retrieveInfo() : String[]
+retrievePhotos() : File[]
LearningAgPage
+uploadLearningAg() : boolean
+submitLearningAg() : File
+generateAndDownload() : File
+generateAndSend() : boolean
ViewUploadedFilesPage
+goToLearningAg()
+goToPreApproval()
+getAllFiles() : File[]
EditProfPage
+uploadResume(resume : File)
+saveChanges()
+changePassword(oldPw : String, newPw : String) : boolean
+changeColor(newColor : int) : boolean
+showResume() : boolean
EditUniPage
+uploadPhotos(photos : File[])
+saveChanges()
+changePassword(oldPw : String, newPw : String) : boolean
+changeColor(newColor : int) : boolean
+editWebsiteURL(url : String) : boolean
+editAcceptedDepartments(departments : String) : boolean
ContactPage
+getExperiencedStudents() : ExperiencedStudents[]
+goToEvaluation()
+goToProfilePage()

Figure 5. User Interface Management Layer

The user interface management layer resides between users and the application layer. This diagram describes how users can interact with the application and what users are capable of. In the diagram above, various user interfaces for different users are represented. In order to distinguish user types from each other, pages that a user has access to are column-wise aligned. For instance, at the leftmost side of the diagram, pages that outgoing students have access to are shown. Related user types are stated at the top of every column.
Higher resolution of this diagram can be found here:
http://elif.kervan.ug.bilkent.edu.tr/BEAM/User_Interface_Management_Layout_v2.jpg

3.3.2. Web Server Layer

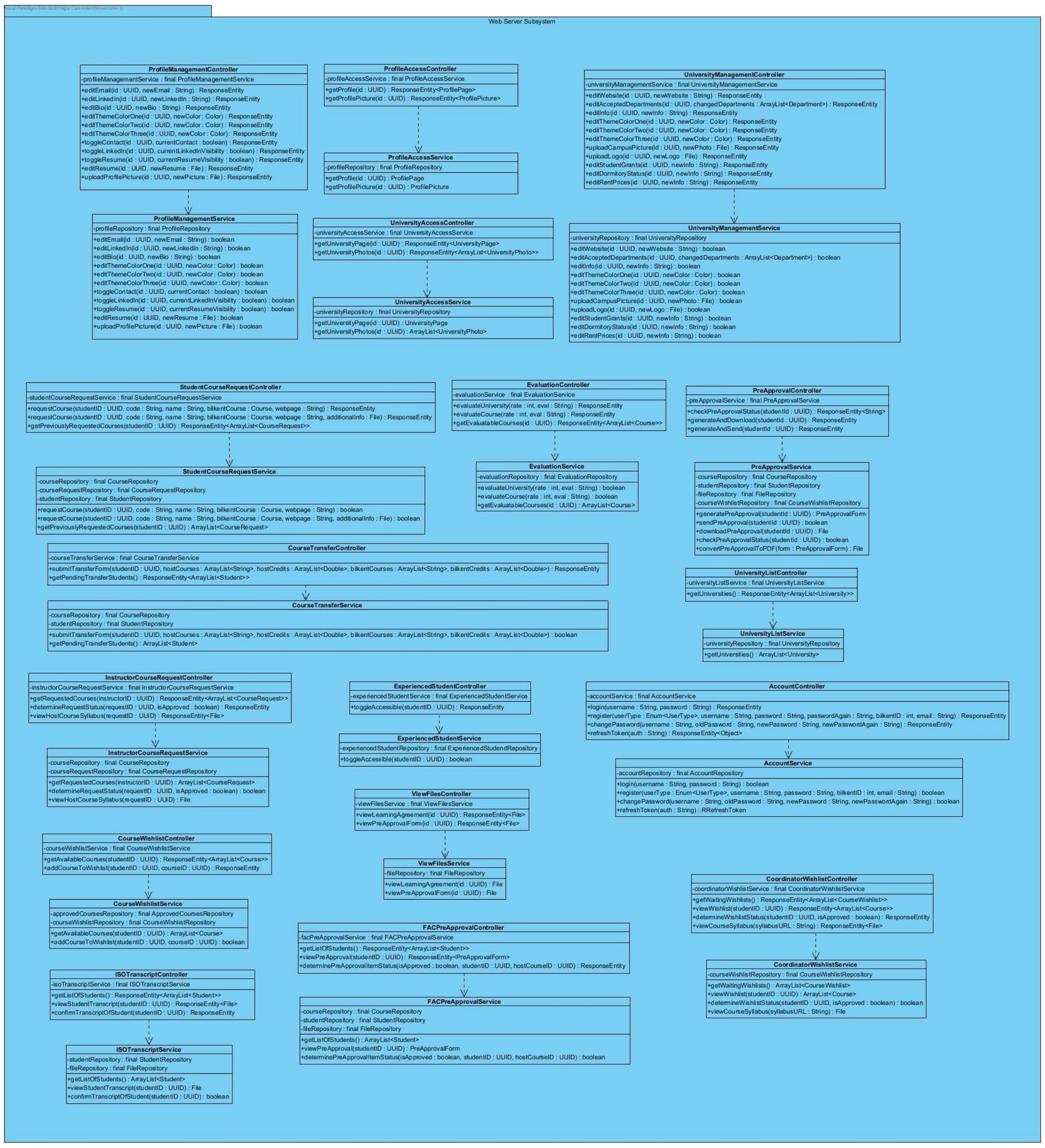


Figure 6. Web Server Subsystem Diagram

Web Server Subsystem diagram shows the endpoint functions, relevant parameters, return types, attributes and relations between the controllers and services (whose designs are inspired from a semi-Microservices architecture). Higher resolution of this diagram can be found here:
http://elif.kervan.ug.bilkent.edu.tr/BEAM/Web-Service-Layer-Iter-2.jpg

3.3.3. Data Management Layer

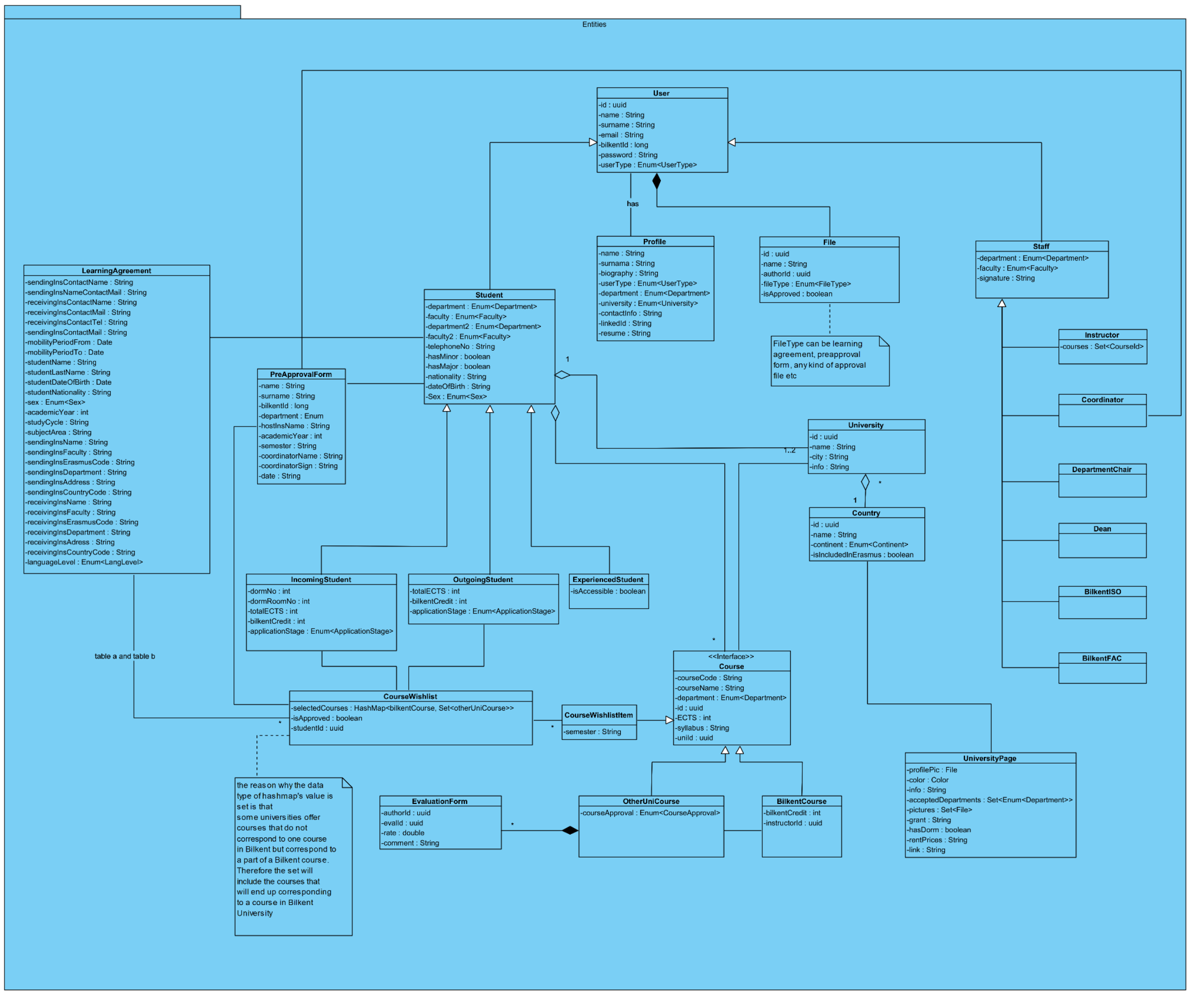


Figure 7. Data Management Layer - Entity Diagram

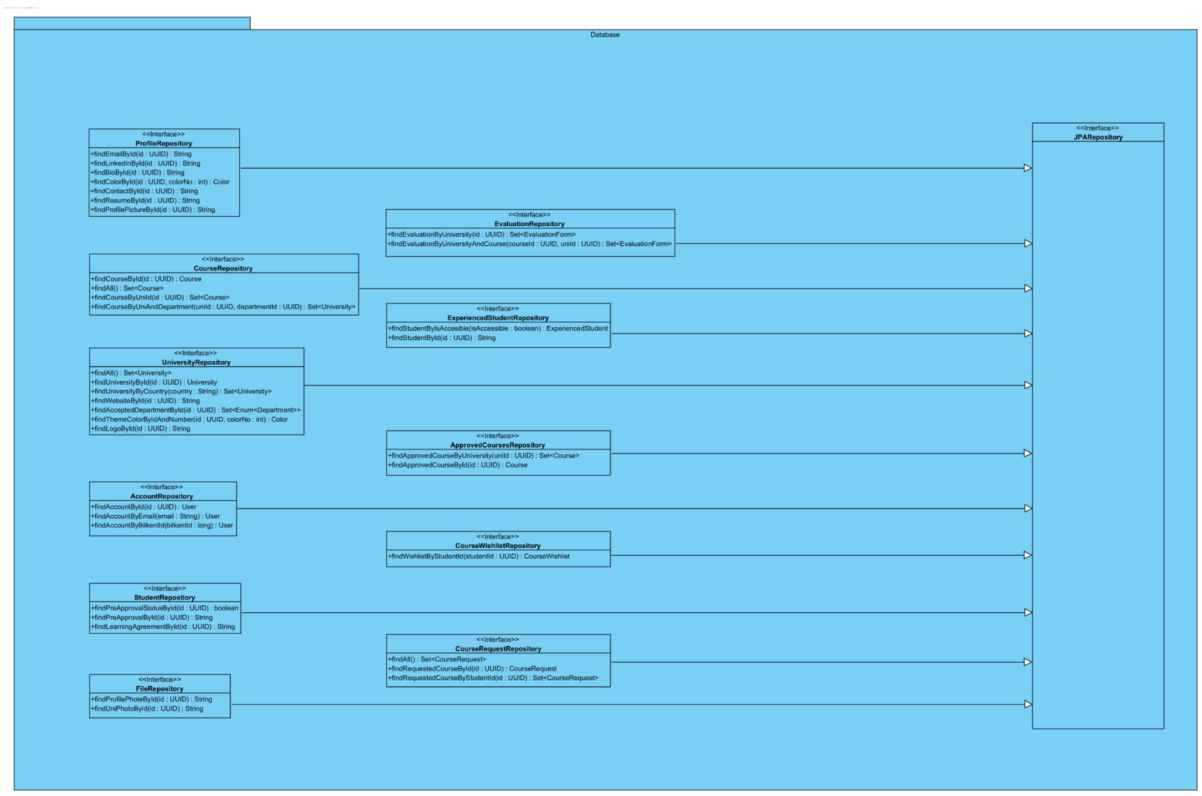

Database
<<Interface>>
ProfileRepository
+findEmailById(id : UUID) : String
+findLinkedInById(id : UUID) : String
+findBioById(id : UUID) : String
+findColorById(id : UUID, colorNo : int) : Color
+findContactById(id : UUID) : String
+findResumeById(id : UUID) : String
+findProfilePictureById(id : UUID) : String
<<Interface>>
EvaluationRepository
+findEvaluationByUniversity(id : UUID) : Set<EvaluationForm>
+findEvaluationByUniversityAndCourse(courseId : UUID, uniId : UUID) : Set<EvaluationForm>
<<Interface>>
CourseRepository
+findCourseById(id : UUID) : Course
+findAll() : Set<Course>
+findCourseByUniId(id : UUID) : Set<Course>
+findCourseByUniAndDepartment(uniId : UUID, departmentId : UUID) : Set<University>
<<Interface>>
ExperiencedStudentRepository
+findStudentByIsAccesible(isAccessible : boolean) : ExperiencedStudent
+findStudentById(id : UUID) : String
<<Interface>>
UniversityRepository
+findAll() : Set<University>
+findUniversityById(id : UUID) : University
+findUniversityByCountry(country : String) : Set<University>
+findWebsiteById(id : UUID) : String
+findAcceptedDepartmentById(id : UUID) : Set<Enum<Department>>
+findThemeColorByIdAndNumber(id : UUID, colorNo : int) : Color
+findLogoById(id : UUID) : String
<<Interface>>
ApprovedCoursesRepository
+findApprovedCourseByUniversity(uniId : UUID) : Set<Course>
+findApprovedCourseById(id : UUID) : Course
<<Interface>>
AccountRepository
+findAccountById(id : UUID) : User
+findAccountByEmail(email : String) : User
+findAccountByBilkentId(bilkentId : long) : User
<<Interface>>
CourseWishlistRepository
+findWishlistByStudentId(studentId : UUID) : CourseWishlist
<<Interface>>
StudentRepostiory
+findPreApprovalStatusById(id : UUID) : boolean
+findPreApprovalById(id : UUID) : String
+findLearningAgreementById(id : UUID) : String
<<Interface>>
CourseRequestRepository
+findAll() : Set<CourseRequest>
+findRequestedCourseById(id : UUID) : CourseRequest
+findRequestedCourseByStudentId(id : UUID) : Set<CourseRequest>
<<Interface>>
FileRepository
+findProfilePhoteById(id : UUID) : String
+findUniPhotoById(id : UUID) : String
<<Interface>>
JPARepository

Figure 8. Data Management Layer - Repository Diagram

Data Management Layer consists of two components, which are Repository and Entity components. Repository layer communicates with the database by sending or retrieving data using the related entities shown in the Entity diagram. Higher resolution of these diagrams can be found here:
http://elif.kervan.ug.bilkent.edu.tr/cs319_design_report/Database_Models.png
http://elif.kervan.ug.bilkent.edu.tr/cs319_design_report/repostitory_diagram.png

3.4. Class Interfaces

3.4.1. User Interface Layer Class Interfaces

3.4.1.1. LoginPage

public loginRequest(String bilkentId, String password): On click, sends a request to the webserver to verify user credentials.
public goToForgotPassword(): Goes to the forgot password page on click

3.4.1.2. RegisterPage

public registerInBulk(List<User> users): On click, registers the list of users provided by the admin in a bulk
public register(User user): On click, registers the user provided by the admin

3.4.1.3. ForgotPwPage

public forgotPasswordRequest(String email): On click, sends a verification email to the user's email
public goToLoginPage(): On click, goes to the login page

3.4.1.4. UniversitiesPage

public searchUni(criterion: String): University[] : When a user enters a search criterion to the text input, the matching universities are searched and displayed
public getUniversities(): University[]: List of all available universities is fetched and displayed.

3.4.1.5. University Details Page

public UniversityDetail getUniversityDetails(universityId: String): Details of the university with the given id is fetched
public seeQuickInformation(): On click, quick information like university name, dormitory info, student grants, city, country is provided.
public seeGeneralInformation(): Upon click, general information about the university is provided.
public seeAcceptanceConditions(): On click, acceptance conditions like language criteria, semester limitations, department limitations are displayed.

3.4.1.6. CourseWishlistPage

public Course[] getAddedCourses():
public Course[] getCoursesThatCanBeAdded():
public boolean addCourse(hostCourse: HostCourse, bilkentCourse: Course):
public boolean removeCourse(courseId: String):

3.4.1.7. CourseRequestPage

public boolean uploadSyllabus(syllabus: File): When clicked, uploads the given syllabus to the system.
public boolean requestCourse(~~preApproval: File~~): On click, sends the course request to the instructor of that course.
public Course[] getPreviouslyRequestedCourses(): Returns the courses that are previously requested by the students.

3.4.1.8. CoursePreAppPage

public boolean uploadPreApproval(preApp: File): Uploads the pre approval form
public boolean submitPreApproval(preApp: File): Submits the pre approval form for the inspection of the coordinator
public File generateAndDownload(): Generates a pre-approval form based on the course wishlist user provided and downloads the generated pre-approval form
public File generateAndSend(): Generates a pre-approval form based on the course wishlist user provided and sends the generated pre-approval form for the inspection of the coordinator

3.4.1.9. LearningAgPage

public boolean uploadLearningAgreement(): Uploads the learning agreement form
public File submitLearningAgreemen(): Submits the uploaded learning agreement form for the inspection of the coordinator
public File generateAndDownload(): Generates a partially filled learning agreement form based on the pre-approval form and downloads the generated learning agreement form.

3.4.1.10. ContactPage

public ExperiencedStudents[] getExperiencedStudents(): On page load, fetches the experienced students
public goToEvaluation(): On click goes to the evaluations made by the experienced student
public goToProfilePage(): On click goes to the profile of the experienced student

3.4.1.11. CourseListPage

public Course[] getCourseList(): On click, gets the available courses in Bilkent

3.4.1.12. ChooseCoursesPage

public addCourse(): On click, adds the chosen course
public removeCourse(): On click, removes the chosen course
public submitCourses(): On click, submits the course list for the approval of the coordinator
public Course [] getCoursesThatCanBeAdded(): On page load, fetches courses that can be added
public Course[] getAddedCourses(): On page load, gets already added courses

3.4.1.13. UniEvalPage

public saveEvaluation(): On click, saves the evaluation for later inspection
public submitEvaluation(): On click submits the university evaluation

3.4.1.14. CourseEvalPage

public selectCourse(): On selection, selects the course which the evaluation will be written for
public Course[] getCoursesTaken(): On page load, fetches all the courses the experienced student has taken in the host university
public saveEvaluation(): On click, saves the evaluation for later inspection
public submitEvaluation(): On click submits the course evaluation

3.4.1.15. ApproveStudentsWishlistPage

public viewAStudentsWishlist(): On click, views the specific wishlist
public approveWishlist(): On click, approves the wishlist
public rejectWishlist(): On click rejects the wishlist
public Wishlist[] getAllWishlists(): On page load gets all of the wishlists

public Course[] getCourses: On page load, gets all the

3.4.1.16. CourseTransferFormPage

public editCourseTransfer(): On change, edits the course transfer form details
public submitTransferForm(): On click, submits the transfer form

3.4.1.17. ApproveLearningAgreementPage

public viewStudentLearningAgreement(): On page load, displays all of the student learning agreements
public approveLearningAgreement(): On click, approves the learning agreement
public rejectLearningAgreement(): On click, rejects the learning agreement

3.4.1.18. ApprovePreApprovalsPage

public chooseStudent(): Chooses the student whom PreApprovalForm will be displayed.
public Student [] getAllStudents(): Fetches all the students related with the co
public approvePreApproval(): On click, approves the pre approval
public rejectPreApproval(): On click, rejects the pre approval
public PreApproval getPreApproval(studentId: long): Upon student selection, fetches the pre approval of the student with the given studentId.

3.4.1.19. ApproveCourseRequestsPage

public approveCourse(courseRequestId: string): Approves the specific course request and any other identical requests for the same university. The approval is logged.
public rejectCourse(instructorId: string): Rejects the specific course request and any other identical requests for the same university. Optionally, provides textual feedback.
public CourseRequest[] getCourseRequests(instructorId: string): Gets all the course requests available for the given instructor

3.4.1.20. TranscriptUploadPage

public selectCoordinator(): Selects the coordinator and fetches relevant student data on click.
public selectStudent(): Selects the student associated with the coordinator.
public sendTranscript(): Sends the transcript to the system, the transcript becomes visible to the coordinator

3.4.1.21. AdminEditPage

public boolean editUniversityInfo(info: String): On click submits the changes made for that university.
public removeAUser(userID: int): Removes the user with the specific id from the system.
public uploadAnyFile(aFile: File, userID: int): Uploads the given file to the user with the given ID.
public removeEvaluation(userID: int): Removes the evaluations of the given user.

3.4.1.22. SideBar

public goToUniversities(): On click, navigates to Universities Page
public goToCourseWishlist(): On click, navigates to Course Wishlist Page
public goToPreApproval(): On click, navigates to PreApproval Page
public goToLearningAgreement(): On click, navigates to Learning Agreement Page
public goToCourseList(): On click, navigates to Course List Page
public goToUniEval(): On click, navigates to University Evaluation Page
public goToCoursesEval(): On click, navigates to Course Evaluation Page
public goToViewUploadedFiles(): On click, navigates to View Uploaded Files Page
public goToStudentWishlists: On click, navigates to Approve Student Wishlists Page
public goToApproveCourses: On click, navigates to Approve Course Request Page
public goToApprovePreApproval: On click, navigates to Approve Pre Approva**l** Page
public goToApproveLearningAgreement: On click, navigates to Approve Learning Agreement Page
public goToCourseTransferForm(): On click, navigates to Course Transfer Form Page
public logout(): On click, logs the user out of the system.

3.4.1.23. NavBar

public goToProfile(): On click, navigates to Profile Page.
public goToNotifications(): On click, navigates to Notifications Page

3.4.1.24. ViewUploadedFilesPage

public goToLearningAgreement(): On click, goes to Learning Agreement Page
public goToPreApproval(): On click, navigates to Pre Approval Page
public File[] getAllFiles(): Gets all the files of the user

3.4.1.25. YourProfPage

public editProfile(): Allows the user to edit their profile
public String[] retrieveInfo(): Gets the information of the current user

3.4.1.26. OthersProfPage

public String[] retrieveInfo(): Gets the information of the selected user

3.4.1.27. EditProfPage

public uploadResume(resume:File): The user uploads their resume to their profile.
public saveChanges(): Saves the changes made by the user
public boolean changePassword(): Changes the user’s password
public boolean showResume(): Adjusts wheter the resume of the user is publicly visible.

3.4.1.28. UniPageByStudent

public String[] retrieveInfo: Gets the information of the university from the system
public File[] retrievePhotos: Gets the photo of the university and displays them

3.4.2. Web Server Layer Class Interfaces

3.4.2.1. ProfileManagementController

Attributes:
private final ProfileManagementSerice profileManagementSerice: This is the Profile Management service for all users

Operations:
public ResponseEntity editMail(UUID id, String newEmail): This operation edits users email information.
public ResponseEntity editLinkedIn(UUID id, String newLinkedIn): This operation edits users LinkedIn information.
public ResponseEntity editBio(UUID id, String newBio): This operation edits users biography.
public ResponseEntity editThemeColorOne(UUID id, Color newColor): This operation edits users color choice one.
public ResponseEntity editThemeColorTwo(UUID id, Color newColor): This operation edits users color choice two.
public ResponseEntity editThemeColorThree(UUID id, Color newColor): This operation edits users color choice three.
public ResponseEntity toggleContact(UUID id, boolean currentContact): This operation changes contact status of a user for the logged user.
public ResponseEntity toggleLinkedIn(UUID id, boolean currentLinkedInVisibility): This operation changes linkedIn visibility status of the logged user.
public ResponseEntity toggleResume(UUID id, boolean currentResumeVisibility): This operation changes resume visibility status of the logged user.
public ResponseEntity editResume(UUID id, File newResume): This operation uploads a new resume file instead of the existing one or uploads for the first time.
public ResponseEntity uploadProfilePicture(UUID id, File newPicture): This operation uploads a new profile picture instead of the existing one or uploads for the first time.

3.4.2.2. ProfileMangementService

Attributes:
private final ProfileRepository profileRepository: This is database repository for profiles

Operations:
public boolean editMail(UUID id, String newEmail): This operation edits users email information.
public boolean editLinkedIn(UUID id, String newLinkedIn): This operation edits users LinkedIn information.
public boolean editBio(UUID id, String newBio): This operation edits users biography.
public boolean editThemeColorOne(UUID id, Color newColor): This operation edits users color choice one.
public boolean editThemeColorTwo(UUID id, Color newColor): This operation edits users color choice two.

public boolean editThemeColorThree(UUID id, Color newColor): This operation edits users color choice three.
public boolean toggleContact(UUID id, boolean currentContact): This operation changes contact status of a user for the logged user.
public boolean toggleLinkedIn(UUID id, boolean currentLinkedInVisibility): This operation changes linkedIn visibility status of the logged user.
public boolean toggleResume(UUID id, boolean currentResumeVisibility): This operation changes resume visibility status of the logged user.
public boolean editResume(UUID id, File newResume): This operation uploads a new resume file instead of the existing one or uploads for the first time.
public boolean uploadProfilePicture(UUID id, File newPicture): This operation uploads a new profile picture instead of the existing one or uploads for the first time.

3.4.2.3. ProfileAccessController

Attributes:
private final ProfileAccessService profileAccessService: This is a service for a user to access other users' profiles.

Operations:
public ResponseEntity<ProfilePage> getProfile(UUID id): This operation retrieves selected profile page based on the id.
public ResponseEntity<ProfilePicture> getProfile(UUID id): This operation retrieves selected profile's profile picture based on the id.

3.4.2.4. ProfileAccessService

Attributes: private final ProfileRepository profileRepository: This is database repository for profiles.

Operations:
public ProfilePage getProfile(UUID id): This operation retrieves selected profile page based on the id.
public ProfilePicture getProfile(UUID id): This operation retrieves selected profile's profile picture based on the id.

3.4.2.5. UniversityAccessController

Attributes:
private final UniversityAccessService universityAccessService: This is a service for users to access universities' pages.

Operations:
public ResponseEntity<UniversityPage> getUniversityPage(UUID id): This operation gets corresponding university page based on its id.
public ResponseEntity<ArrayList<UniversityPage>> getUniversityPhotos(UUID id): This operation gets corresponding university's photos based on its id.

3.4.2.6. UniversityAccessService

Attributes:
private final UniversityRepository universityRepository: This is a database repository for universities.

Operations:
public UniversityPage getUniversityPage(UUID id): This operation gets corresponding university page based on its id.
public UniversityPage getUniversityPhotos(UUID id): This operation gets corresponding university’s photos based on its id.

3.4.2.7. UniversityManagementController

Attributes:
private final UniversityManagementService universityManagementService: This is a service for universities to manages/edits their profiles.

Operations:
public ResponseEntity editWebsite(UUID id, String newWebsite): This operation edits saved URL information of the corresponding university.
public ResponseEntity editAcceptedDepartments(UUID id,ArrayList<Department> changedDepartments): This operation edits saved accepted departments information of the corresponding university.
public ResponseEntity editInfo(UUID id, String newInfo): This operation edits saved general information of the corresponding university.
public ResponseEntity editThemeColorOne(UUID id, Color newColor): This operation edits universities’ color choice one.
public ResponseEntity editThemeColorTwo(UUID id, Color newColor): This operation edits universities’ color choice two.
public ResponseEntity editThemeColorThree(UUID id, Color newColor): This operation edits universities’ color choice three.
public ResponseEntity uploadCampusPicture(UUID id, File newPhoto): This operation uploads campus photos to universities’ pages.
public ResponseEntity uploadLogo(UUID id, File newLogo): This operation uploads a new logo to universities’ pages.
public ResponseEntity editStudentGrants(UUID id, String newInfo): This operation uploads student grant information of the corresponding university.
public ResponseEntity editDormitoryStatus(UUID id, String newInfo): This operation uploads dormitory information of the corresponding university.
public ResponseEntity editRentPrices(UUID id, String newInfo): This operation uploads rent prices information of the corresponding university.

3.4.2.8. UniversityManagementService

Attributes:
private final UniversityRepository universityRepository: This is a database repository for universities.

Operations:
public boolean editWebsite(UUID id, String newWebsite): This operation edits saved URL information of the corresponding university.
public boolean editAcceptedDepartments(UUID id,ArrayList<Department> changedDepartments): This operation edits saved accepted departments information of the corresponding university.
public boolean editInfo(UUID id, String newInfo): This operation edits saved general information of the corresponding university.
public boolean editThemeColorOne(UUID id, Color newColor): This operation edits universities' color choice one.
public boolean editThemeColorTwo(UUID id, Color newColor): This operation edits universities' color choice two.
public boolean editThemeColorThree(UUID id, Color newColor): This operation edits universities' color choice three.
public boolean uploadCampusPicture(UUID id, File newPhoto): This operation uploads campus photos to universities' pages.
public boolean uploadLogo(UUID id, File newLogo): This operation uploads a new logo to universities' pages.
public boolean editStudentGrants(UUID id, String newInfo): This operation uploads student grant information of the corresponding university.
public boolean editDormitoryStatus(UUID id, String newInfo): This operation uploads dormitory information of the corresponding university.
public boolean editRentPrices(UUID id, String newInfo): This operation uploads rent prices information of the corresponding university.

3.4.2.9. StudentCourseRequestController

Attributes:
private final StudentCourseRequestService studentCourseRequestService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for students' course request functionality.

Operations:
public ResponseEntity requestCourse(UUID studentID, String code, String name, Course bilkentCourse, String webpage): This operation gathers related data from a RequestBody and makes a course request for a student via the aforementioned service.

public ResponseEntity requestCourse(UUID studentID, String code, String name, Course bilkentCourse, String webpage, File additionalInfo): This operation gathers related data from a RequestBody and makes a course request for a student via the aforementioned service.
public ResponseEntity<ArrayList<CourseRequest>> getPreviouslyRequestedCourses(UUID studentId): This operation gives the previously requested courses of the student.

3.4.2.10. StudentCourseRequestService

Attributes:
private final CourseRepository courseRepository: This repository holds information about courses.
private final CourseRequestRepository courseRequestRepository: This repository holds information about course requests.
private final StudentRepository studentRepository: This repository holds information about students.

Operations:
public boolean requestCourse(UUID studentID, String code, String name, Course bilkentCourse, String webpage): This operation makes a course request for a student by gathering data from the databases given above and saves the request to the courseRequestRepository repository.
public boolean requestCourse(UUID studentID, String code, String name, Course bilkentCourse, String webpage, File additionalInfo): This operation makes a course request for a student by gathering data from the databases given above and saves the request to the courseRequestRepository repository.
public ArrayList<CourseRequest> getPreviouslyRequestedCourses(UUID studentId): This operation gives the previously requested courses of the student by accessing the courseRequestRepository repository.

3.4.2.11. EvaluationController

Attributes:
-evaluationService : final EvaluationService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for course/university evaluation related functionalities.

Operations:
+evaluateUniversity(rate : int, eval : String) : ResponseEntity: This operation rates a University and stores the evaluation input.
+evaluateCourse(rate : int, eval : String) : ResponseEntity: This operation rates a Course in a University and stores the evaluation input.
+getEvaluatableCourses(id : UUID) : ResponseEntity<ArrayList<Course>>: This operation returns all available courses to be evaluated by the student.

3.4.2.12. EvaluationService

Attributes:

-evaluationRepository : final EvaluationRepository: This repository holds information about past evaluations.

Operations:
+evaluateUniversity(rate : int, eval : String) : boolean: This operation rates a University and stores the evaluation input.
+evaluateCourse(rate : int, eval : String) : boolean: This operation rates a Course in a University and stores the evaluation input.
+getEvaluatableCourses(id : UUID) : ArrayList<Course>: This operation returns all available courses to be evaluated by the student.

3.4.2.13. PreApprovalController

Attributes:
-preApprovalService : final PreApprovalService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for PreApproval related functionalities like generation and downloading.

Operations:
+checkPreApprovalStatus(studentId : UUID) : ResponseEntity<String>: This operation checks the PreApproval “approval” status of a student and gives relevant information like accepted/rejected.
+generateAndDownload(studentId : UUID) : ResponseEntity: This operation generates a PreApproval form from BEAM’s systems and returns relevant information for a downloadable PDF of the PreApproval form.
+generateAndSend(studentId : UUID) : ResponseEntity: This operation generates a PreApproval form from BEAM’s systems and sends the PreApproval form for approval to the relevant authority.

3.4.2.14. PreApprovalService

Attributes:
-courseRepository : final CourseRepository: This repository holds information about courses.
-studentRepository : final StudentRepository: This repository holds information about students.
-fileRepository : final FileRepository: This repository holds information about files uploaded to the system of BEAM.
-courseWishlistRepository : final CourseWishlistRepository: This repository holds information about Courses in a student’s CourseWishlist.

Operations:
+generatePreApproval(studentId : UUID) : PreApprovalForm: This operation generates a PreApproval form from BEAM’s systems.
+sendPreApproval(studentId : UUID) : boolean: This operation sends the PreApproval form for approval to the relevant authority.
+downloadPreApproval(studentId : UUID) : File: This operation returns relevant information for a downloadable PDF of the PreApproval form.

+checkPreApprovalStatus(studentId : UUID) : boolean: This operation checks the PreApproval “approval” status of a student and gives relevant information like accepted/rejected.
+convertPreApprovalToPDF(form : PreApprovalForm) : File: This operation converts the internal representation of a PreApproval form to a PreApproval form in the PDF format.

3.4.2.15. CourseTransferController

Attributes:
-courseTransferService : final CourseTransferService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for course transfer functionality.

Operations:
+submitTransferForm(studentID : UUID, hostCourses : ArrayList<String>, hostCredits : ArrayList<Double>, bilkentCourses : ArrayList<String>, bilkentCredits : ArrayList<Double>) : ResponseEntity: This operation submits a transfer form for the mobility period of a student by using the relevant data in its parameters.
+getPendingTransferStudents() : ResponseEntity<ArrayList<Student>>: This operation returns the remaining transfer students, who still need transfer forms to be submitted.

3.4.2.16. CourseTransferService

Attributes:
-courseRepository : final CourseRepository: This repository holds information about courses.
-studentRepository : final StudentRepository: This repository holds information about students.

Operations:
+submitTransferForm(studentID : UUID, hostCourses : ArrayList<String>, hostCredits : ArrayList<Double>, bilkentCourses : ArrayList<String>, bilkentCredits : ArrayList<Double>) : boolean: This operation submits a transfer form for the mobility period of a student by using the relevant data in its parameters.
+getPendingTransferStudents() : ArrayList<Student>: This operation returns the remaining transfer students, who still need transfer forms to be submitted.

3.4.2.17. UniversityListController

Attributes:
-universityListService : final UniversityListService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for listing all universities in BEAM’s system to users.

Operations:
+getUniversities() : ResponseEntity<ArrayList<University>>: This operation returns all universities in BEAM’s system to users and allows access to each university’s university page.

3.4.2.18. UniversityListService

Attributes:
-universityRepository : final UniversityRepository: This repository holds information about universities.

Operations:
+getUniversities() : ArrayList<University>: This operation returns all universities in BEAM's system to users and allows access to each university's university page.

3.4.2.19. InstructorCourseRequestController

Attributes:
-instructorCourseRequestService : final InstructorCourseRequestService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for the functionalities related to instructors and their reviews of course requests.

Operations:
+getRequestedCourses(instructorID : UUID) : ResponseEntity<ArrayList<CourseRequest>>: This operation returns all course requests for the related instructor's Bilkent courses).
+determineRequestStatus(requestID : UUID, isApproved : boolean) : ResponseEntity: This operation approves or rejects a specific course request.
+viewHostCourseSyllabus(requestID : UUID) : ResponseEntity<File>: This operation returns the corresponding host course's syllabus (in a course request) for the instructor to see.

3.4.2.20. InstructorCourseRequestService

Attributes:
-courseRepository : final CourseRepository: This repository holds information about courses.
-courseRequestRepository : final CourseRequestRepository: This repository holds information about course requests of students.

Operations:
+getRequestedCourses(instructorID : UUID) : ArrayList<CourseRequest>: This operation returns all course requests for the related instructor's Bilkent courses).
+determineRequestStatus(requestID : UUID, isApproved : boolean) : boolean: This operation approves or rejects a specific course request.
+viewHostCourseSyllabus(requestID : UUID) : File: This operation returns the corresponding host course's syllabus (in a course request) for the instructor to see.

3.4.2.21. ExperiencedStudentController

Attributes:

-experiencedStudentService : final ExperiencedStudentService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for toggling ExperiencedStudents' communication preference(s).

Operations:
+toggleAccessible(studentID : UUID) : ResponseEntity: This operation toggles (if true, sets to false and vice-versa) the accessibility (in terms of communication) settings of experienced students.

3.4.2.22. ExperiencedStudentService

Attributes:
-experiencedStudentRepository : final ExperiencedStudendRepository: This repository holds information about experienced students who had Erasmus experience at some point in the past.

Operations:
+toggleAccessible(studentID : UUID) : boolean: This operation toggles (if true, sets to false and vice-versa) the accessibility (in terms of communication) settings of experienced students.

3.4.2.23. AccountController

Attributes:
-accountService : final AccountService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for functionalities related to JWT, authentication and account information.

Operations:
+login(username : String, password : String) : ResponseEntity: This operation is used for a user to login.
+register(userType : Enum<UserType>, username : String, password : String, passwordAgain : String, bilkentID : int, email : String) : ResponseEntity: This operation is used for a user to register to BEAM.
+changePassword(username : String, oldPassword : String, newPassword : String, newPasswordAgain : String) : ResponseEntity: This operation is used for a user to change their previous password to a new password.
+refreshToken(auth : String) : ResponseEntity<Object>: This operation is used to give a refreshed JWT token to the user based on their authentication information.

3.4.2.24. AccountService

Attributes:
-accountRepository : final AccountRepository: This repository holds information about accounts like password hashes and usernames.

Operations:
+login(username : String, password : String) : boolean: This operation is used for a user to login.

+register(userType : Enum<UserType>, username : String, password : String, bilkentID : int, email : String) : boolean: This operation is used for a user to register to BEAM.
+changePassword(username : String, oldPassword : String, newPassword : String, newPasswordAgain : String) : boolean: This operation is used for a user to change their previous password to a new password.
+refreshToken(auth : String) : RRefreshToken: This operation is used to give a refreshed JWT token to the user based on their authentication information.

3.4.2.25. ViewFilesController

Attributes:
-viewFilesService : final ViewFilesService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for functionalities related to checking uploaded files to BEAM's systems.

Operations:
+viewLearningAgreement(id : UUID) : ResponseEntity<File>: This operation returns relevant information for a downloadable PDF of the Learning Agreement for students to see.
+viewPreApprovalForm(id : UUID) : ResponseEntity<File>: This operation returns relevant information for a downloadable PDF of the PreApproval form for students to see.

3.4.2.26. ViewFilesService

Attributes:
-fileRepository : final FileRepository: This repository holds information about files uploaded to the system of BEAM.

Operations:
+viewLearningAgreement(id : UUID) : File: This operation returns relevant information for a downloadable PDF of the Learning Agreement for students to see.
+viewPreApprovalForm(id : UUID) : File: This operation returns relevant information for a downloadable PDF of the PreApproval form for students to see.

3.4.2.27. CourseWishlistController

Attributes:
-courseWishlistService : final CourseWishlistService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for functionalities related to coordinators approving/rejecting student course wishlists.

Operations:
+getAvailableCourses(studentID : UUID) : ResponseEntity<ArrayList<Course>>: This operation returns all available courses for a specific student to add their course wishlist.
+addCourseToWishlist(studentID : UUID, courseID : UUID) : ResponseEntity: This operation adds a specific course to a specific student's course wishlist.

3.4.2.28. CourseWishlistService

Attributes:
-approvedCoursesRepository : final ApprovedCoursesRepository: This repository holds information about previously approved courses.
-courseWishlistRepository : final CourseWishlistRepository: This repository holds information about Courses in a student's CourseWishlist.

Operations:
+getAvailableCourses(studentID : UUID) : ArrayList<Course>: This operation returns all available courses for a specific student to add their course wishlist.
+addCourseToWishlist(studentID : UUID, courseID : UUID) : boolean: This operation adds a specific course to a specific student's course wishlist.

3.4.2.29. ISOTranscriptController

Attributes:
-isoTranscriptService : final ISOTranscriptService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for the functionalities related to student transcript.

Operations:
+getListOfStudents(): This operation returns all of the students that has a transcript.
viewStudentTranscript(UUID studentId): This operation returns the transcript of the student with the given student id.
+confirmTranscriptOfStudent(UUID studentId): This operation confirms the transcript of the selected student.

3.4.2.30. ISOTranscriptService

Attributes:
-studentRepository : final StudentRepository: This repository holds information about students.
-fileRepository : final FileRepository: This repository holds information about files uploaded to the system of BEAM.

Operations:
+getListOfStudents(): This operation returns all of the students that has transcript.
viewStudentTranscript(UUID studentId): This operation returns the transcript of the student with the given student id.
+confirmTranscriptOfStudent(UUID studentId): This operation confirms the transcript of the selected student.

3.4.2.31. FACPreApprovalController

Attributes:
-facPreApprovalService : final FACPreApprovalService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for the functionalities related to Pre-Approval Form.

Operations:
+getListOfStudents(): This operation returns all of the students that submitted Pre-Approval Form**.**
+viewPreApproval(UUID studentId): This operation returns the Pre-Approval Form of the selected student.
+determinePreApprovalItemStatus(boolean isApproved, UUID studentId, UUID hostCourseId): This operation is used to approve or reject a pre approval item of the selected student.

3.4.2.32. FACPreApprovalService

Attributes:
-courseRepository : final CourseRepository: This repository holds information about courses.
-studentRepository : final StudentRepository: This repository holds information about students.
-fileRepository : final FileRepository: This repository holds information about files uploaded to the system of BEAM.

Operations:
+getListOfStudents(): This operation returns all of the students that submitted Pre-Approval Form**.**
+viewPreApproval(UUID studentId): This operation returns the Pre-Approval Form of the selected student.
+determinePreApprovalItemStatus(boolean isApproved, UUID studentId, UUID hostCourseId): This operation is used to approve or reject a pre approval item of the selected student.

3.4.2.33. CoordinatorWishlistController

Attributes:
-coordinatorWishlistService : final CoordinatorWishlistService: This service gives the controller class the ability to access business-related functions and CRUD operations on the related database(s) for the functionalities related to student wishlist.

Operations:
+getWaitingWishlists() : ResponseEntity<ArrayList<CourseWishlist>>: This operation returns all pending wishlists to be approved/rejected.
+viewWishlist(studentID : UUID) : ResponseEntity<ArrayList<Course>>: This operation returns the course wishlist information of a specific student.
+determineWishlistStatus(studentID : UUID, isApproved : boolean) : ResponseEntity: This operation approves or rejects a student's course wishlist.

+viewCourseSyllabus(syllabusURL : String) : ResponseEntity<File>: This operation returns the course's syllabus (for a course which is in a student's course wishlist) for a coordinator to see.

3.4.2.34. CoordinatorWishlistService

Attributes:
-courseWishlistRepository : final CourseWishlistRepository: This repository holds information about Courses in a student's CourseWishlist.

Operations:
+getWaitingWishlists() : <CourseWishlist>: This operation returns all pending wishlists to be approved/rejected.
+viewWishlist(studentID : UUID) : ArrayList<Course>: This operation returns the course wishlist information of a specific student.
+determineWishlistStatus(studentID : UUID, isApproved : boolean) : boolean: This operation approves or rejects a student's course wishlist.
+viewCourseSyllabus(syllabusURL : String) : File: This operation returns the course's syllabus (for a course which is in a student's course wishlist) for a coordinator to see.

3.4.3. Data Management Layer Class Interfaces

3.4.3.1. ProfileRepository

Operations:
findEmailById(id:UUID): Finds user email from their uuid.
findLinkedInById(id:UUID): Finds user LinkedIn profile link from their uuid.
findBioById(id:UUID): Finds user biography from their uuid.
findLinkedInById(id:UUID): Finds user LinkedIn profile link from their uuid.
findColorById(id:UUID, colorNo: int): Finds the color theme of a user's profile page using the theme color number and user uuid.
findContactById(id:UUID): Finds user contact from their uuid.
findResumeById(id:UUID): Finds user resume from their uuid.
findProfilePictureById(id:UUID): Finds user profile picture from their uuid.

3.4.3.2. CourseRepository

Operations:
findCourseById(id:UUID): Finds the course with the given uuid.
findAll(): Finds all of the courses in the database.
findCourseByUniId(id:UUID): Finds the courses in a university using the given university uuid.
findCourseByUniAndDepartment(uniId:UUID, departmentId): Finds the courses in the selected university's selected department using university and department uuid.

3.4.3.3. UniversityRepository

Operations:

findAll(): Finds all of the universities in the database.

findUniversityByCountry(country:String): Finds the universities in the selected country using the country name.
findWebsiteById(id:UUID): Finds the website link of the university with the given uuid.
findAcceptedDepartmentById(id:UUID): Finds the accepted departments in the selected university with the given uuid.
findThemeColorByIdAndNumber(id:UUID, colorNo: int): Finds the theme color of the selected university with the given uuid and color theme number.
findLogoById(id:UUID): Finds the logo of the university with the given university uuid.

3.4.3.4. AccountRepository

Operations:

findAccountById(id:UUID): Finds the user account with the given uuid.
findAccountByEmail(email:String): Finds the account with the given email.
findCourseByBilkentId(bilkentId:long): Finds the user account with the given Bilkent Id.

3.4.3.5. StudentRepository

Operations:

findPreApprovalStatusById(id:UUID): Finds the current status (approved, pending, or rejected) of the Pre Approval File with the given student uuid.
findPreApprovalById(id:UUID): Finds the Pre Approval File of the given student using student uuid.
findLearningAgreementById(id:UUID): Finds the Learning Agreement of the given student.

3.4.3.6. FileRepository

Operations:

findProfilePhotoById(id:UUID): Finds the profile photo with the given user uuid.

findUniPhotoById(id:UUID): Finds the university photos with the given university uuid.

3.4.3.7. EvaluationRepository

Operations:

findEvaluationByUniversity(id:UUID): Finds the university evaluations with the given university uuid.

findEvaluationByUniversityAndCourse(courseId:UUID, uniId: UUID): Finds the course evaluation of the given university using their uuid.

3.4.3.8. ExperiencedStudentRepository

Operations:

findStudentByIsAccessible(isAccessible:boolean): Finds the experienced students who are willing to help other students.

findStudentById(id:UUID): Finds the experienced student with the given user uuid.

3.4.3.9. ApprovedCoursesRepository

Operations:

findApprovedCourseByUniversity(uniID: uuid): Finds the approved courses of the given university.

findApprovedCourseById(id: UUID): Finds the approved course with the given course uuid.

3.4.3.10. CourseWishlistRepository

Operations:

findWishlistByStudentId(studentId:UUID): Finds the wishlist of the given student.

3.4.3.11. CourseRequestRepository

Operations:

findAll(): Finds all of the requested courses.

findRequestedCourseById(id:uuid): Finds the requested course using the given uuid.

findRequestedCourseByStudentId(studentId:UUID): Finds the requested courses of the given student.

3.5. Design Patterns

3.5.1. Decorator Pattern

Beam handles a lot of file upload and fetching operations. Various application users will need to upload files, access their files when they want to, or override the existing uploaded files. The names of these files should be unique, and the name must point to the user who uploaded the file. To ensure these requirements are met, we decided that using the decorator pattern for the namings of the files that we upload to AWS S3 is essential.

Users may upload the files with whatever naming they want. However, as mentioned before, we need to name the files uniquely while we are storing them in AWS S3. The decorator pattern will provide the following functionalities while we are renaming the files: add user-specific information like student id; add the type of the form as a prefix; add the date of uploading as a postfix. For example, a user might upload the file as “erasmus_preapproval.” Using the decorator pattern, we will manipulate the name, which will become “Pre-Approval_<studentId>_2022-12-11 12:00:00”.

The name of the decorator classes will be Timestamped, FileTyped, UserSpecified. Due to the nature of the decorator pattern, these will be used with any combination possible, and any additional decorations will be made with ease.

3.5.2. Strategy Pattern

Erasmus application process highly depends on the approval/rejection of the authorities for different files and requests such as course requests, Pre-Approval Forms, and Learning Agreements. Furthermore, some of the approvals are completed with a signature, whereas some do not need any other data. In order to provide a maintainable approach for approving or rejecting files/requests with different methods, we will use Strategy Design Pattern. Therefore, we will have 2 strategies; one will be approving with a signature, and the other will be plain approve. Since many types of staff including instructor and coordinator approve some sort of file/request, using the Strategy Pattern will help us maintain the approval process easily. At the same time, it will enable us to add more approval methods in the feature if we need any.

Also, we will be using this pattern during the file upload process. BEAM provides automatic file generation for the documents mentioned above. When students generate a file there are two choices: the first one is downloading the generated document and then submitting it whenever they want, and the second one is submitting the file right after the generation. Since we have two different strategies, we will use Strategy Pattern to implement file uploading process. Thanks to this pattern, we can add more strategies to the file upload system in the future easily.

3.5.3. Façade Pattern

We implement the Facade pattern at two specific locations. One on the web-server layer, and one on the database abstraction layer. On the web-server layer, we have separate structures called services. Services abstract the implementation of the requests on each endpoint. This way, controllers will still work with, at most, a minimal change if the implementation changes.

On the other hand, we have abstracted the database layer using repositories. Repositories handle the communication with the database. Therefore, services do not implement any logic related to the database. This way, changes in the repositories do not affect the services layer.

4. Glossary

Incoming student: Student visiting Bilkent University as part of Erasmus program

Outgoing student: Student visiting an abroad university as part of Erasmus program

Experienced student: Student who completed their Erasmus/Exchange program

Pre-Approval form: Official form that documents the course selections by the student

Learning Agreement: Document required by Erasmus aiming to prepare an efficient exchange process. [3]

Transfer form: Form that documents the transferable courses the student have taken in the host university

Host university: University that the outgoing student will study as a part of Erasmus program

Mobility period: Period that starts at the host university

Pre-mobility period: Period before the student goes to the host university

5. References

[1] "Amazon EC2 Cloud Server Types," AWS, [Online]: https://aws.amazon.com/tr/ec2/instance-types/ [Accessed: 29-Nov-2022].
[2] "Amazon RDS Instance Types," AWS, [Online]: https://aws.amazon.com/rds/instance-types/ [Accessed: 29- Nov- 2022].
[3] "Why use PostgreSQL as a Database for my Next Project in 2022,"Michael Borozenets, [Online]: https://fulcrum.rocks/blog/why-use-postgresql-database [Accessed: 29- Nov- 2022].