Bilkent University Department of Computer Engineering

Senior Design Project

Project Name: Cronus

Low Level Design Report

Group Members: Gizem Karal, Gökberk Boz, Hüseyin Fatih Karahan, Irmak Çeliker

Supervisor: İbrahim Körpeoğlu

Jury Members: Shervin Arashloo and Hamdi Dibeklioglu

Low Level Design Report Feb 28, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492.

Table Of Contents

1. Introduction

Project management is a very significant step during the creation of any kind of product. Especially in team projects, it can be very challenging to keep track of the project, distribute and order the tasks due to the number of people working in the project. Along with the fact that it can be hard to carry out a project with many different developers, due to the COVID-19[1] pandemic it has become even more complicated to keep track of others' work since project management is also needed to be carried out on digital platforms. Our aim is to solve these kinds of complications and give relief to project developers during their project creation process by creating a project management tool, Cronus, for software developers.

Moreover, besides the complications that arise due to the number of members in a project, it can also be challenging for people to keep track of their own process and work. Cronus will eliminate these problems and create a neat work environment for their users.

Every project in addition to normal tasks also has some repetitive tasks that need to be tracked. While Kanban[2] tools like Jira[3] or Asana[4] allow the tracking and ordering of issues, they become cumbersome very quickly when dealing with tasks that need to be repeated over intervals. Usually this is dealt with by creating a new issue during each sprint or whenever it’s needed. Our project management tool instead focuses on the management of tasks that repeat.

1.1 Object design trade-offs

1.1.1 Functionality vs Usability

The two most important aspects for Cronus as a design trade-offs are usability and functionality. Functionality level measures if the functionality of the properties of the software works properly. Usability of Cronus refers to the ease of use and the quality of the users’ experiences. The main group of the users of Cronus app is to be a member of a project which is in a well manageable level. That's why creating a simple and understandable interface for the management of a project is the most significant point for us. Therefore, it must be usable by the users in order to use the application in an efficient and easy way for their projects to be managed and processed in a neat, usable environment. Also, as another consideration the functionality which makes the setting of the general functional points in order to manage the projects’ processes is significant. With proper functionality, to the members of a project, there are assigned tasks by a project leader and everyone can follow the repetitive tasks in the project. It means that the main purpose of the system is to provide a neat, usable and functional platform to the users.

1.1.2 Extensibility & Modularity vs Performance

The extensibility of an application is pretty important in order to extend new properties and upgrade the current system into a higher level. The first versions of every application are different from their upgraded versions, that is why it is important to make a design that is functional in modularity and extensible for any aspect. Without an extensible modularity, it is costly to upgrade the system and extend it.

Each of the packages in the system are implemented independently from each other in order to be changed easily. When there is a dependency among the packages, it is costly to make the needed changes in the dependent packages. it makes the overall system modular when implemented independently.

Modularity helps the system to make changes in specific packages without affecting the other packages as much as possible. However, some performance Hungary packages might affect the overall performance while trying to manage the modularity and excessiveness. More modules and independent ones may lead to some level of consuming performance.

1.1.3 Space vs Time

One of the main features of Cronus is to work simultaneously. The data transfer between the clients with notifications of the project process should last less than a second so that the team members are notified quickly by the changes. Moreover, it is important to control the deadlines in a fast way that with a slow processing time of the application, it can lead to deadline misses and misunderstandings. However, in order to maintain time efficiency, there is an inevitable trade-off between space and time. Our primary focus is to provide an efficient application in the sense of time.

1.1.4 Robustness vs Cost

The people who will benefit from Cronus are aiming to use the app in an easy way without dealing with any problems or errors occurring by the system. They are already dealing with the difficulty of the projects so they should not waste any time by trying to fix the issues. Therefore, our main focus is to provide a system that is robust and reliable against the errors that may occur during the functionality. our choice is to use better services despite our cost. For example, in order to store the information about the project process, it is important to use a reliable database to store the clients information and their tasks. The reliability of the software with its database is much more important than using a cheaper one. Despite the choices of cheap services, it is more significant to choose the expensive one with a more secure one.

1.1.5 Compatibility vs Programmability

In the implementation of the app, we decided to use C# as a language in the ASP.NET platform. As mentioned in the previous reports, the main purpose of the system is to provide a project management tool for the tasks in a project. The app is a web application that is compatible with most of the technologies and we decided to use .NET for that. It is harder to program in C# for the programmability perspective but from the compatibility of the application, it is easier to manage in the web platform.

1.2 Interface documentation guidelines

Class	Class Name
Class Description	
Attributes	
Attribute1	Type - Attribute Description
Attribute2	Type - Attribute Description
Attribute3	Type - Attribute Description
Methods	
method1(param1,param2)	Method description
method1(param1,param2)	Method description

Table 1: Interface Documentation Guidelines

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

In our reports, we based on the Unified Modeling Language (UML) standard [4] for our developers to specify, visualize, construct, and document the artifacts of software systems for modeling class interfaces and interactions and to draw object, entity, package and deployment diagrams and sequence diagrams. References are made using the IEEE standard [5].

1.4 Definitions, acronyms, and abbreviations

App: Stands for ‘application’. An application is a web application platform that is designed for various programming languages and frameworks.

Cronus: It is the name of our application.

Project Management Platform: It is a part of our application which makes the arrangement of the tasks among the users.

Repetitive Tasks: Repetitive tasks are the issues that are repeated in the project more than one.

Account: The personal data of a user such as completed and assigned tasks, their username, password, the project and the team that the member is a part of. An account can be created and be logged into inside the Cronus app.

Issue: Issue is named for the tasks in the project.

Deadline: For the coming tasks, there are deadlines, in other words due dates, which may be completed or changed. By notification the users are notified for the coming deadlines.

Team: It is used for the gathering group of the members that have the same projects. The team has their own groups that can follow their common projects.

Client: A user of the application. So, the client side of Cronus is the user interface side where a user can manage tasks or complete the given tasks.

Profile: It is defined as the personal information grouped together to give the sufficient knowledge about the users.

Project: This abbreviation defines the projects in which the members are a part of. Inside the projects, the assigned tasks are present.

Task: This is the assigned work to be done in the projects as an issue. The tasks can be created and assigned to the users according to the manager’s choice.

Calendar: This is the table of dates in which each user’s proposed and assigned tasks can be seen according to the teams that person belongs to. So, in the calendar view, the calendar belonging to the client, events, meetings, and important dates are presented.

API: It is the abbreviation for the term Application Programming Interface. For the application component, it can be used as an interface in order to make the process easier by the development team.

UML: Unified Modeling Language. A standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems[5].

Server: The back-end of the application. The information about the projects, teams and tasks are kept here.

2. Packages

2.1 Database and Entities

In this part, the application database system can be seen. The application has many type of entities and through redis methods, it uploads those entities according to their identity.

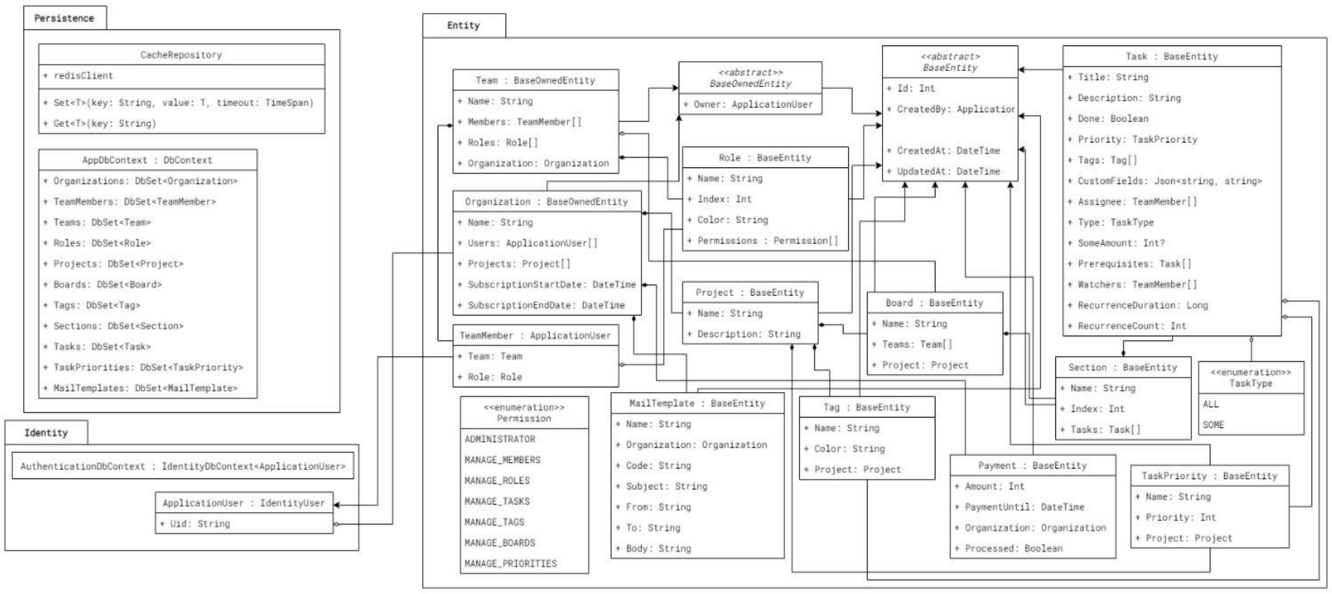

Figure 1: Database and Entity Packages

Persistence

This package contains repositories for communicating with the PostgreSQL database and Redis. DbContext and DbSet classes are from EntityFramework.

Entities

This package contains entities used in the application.

Identity

This package is the bridge between Firebase Authentication and Cronus. ASP.NET Identity will handle user management on the Cronus' side.

2.2 Controller and Service

Controller and service packages are using nearly the same architecture. Every feature of the program like team/member/role or task/task priority or hierarchy of task/project/board are controlled and organized here.

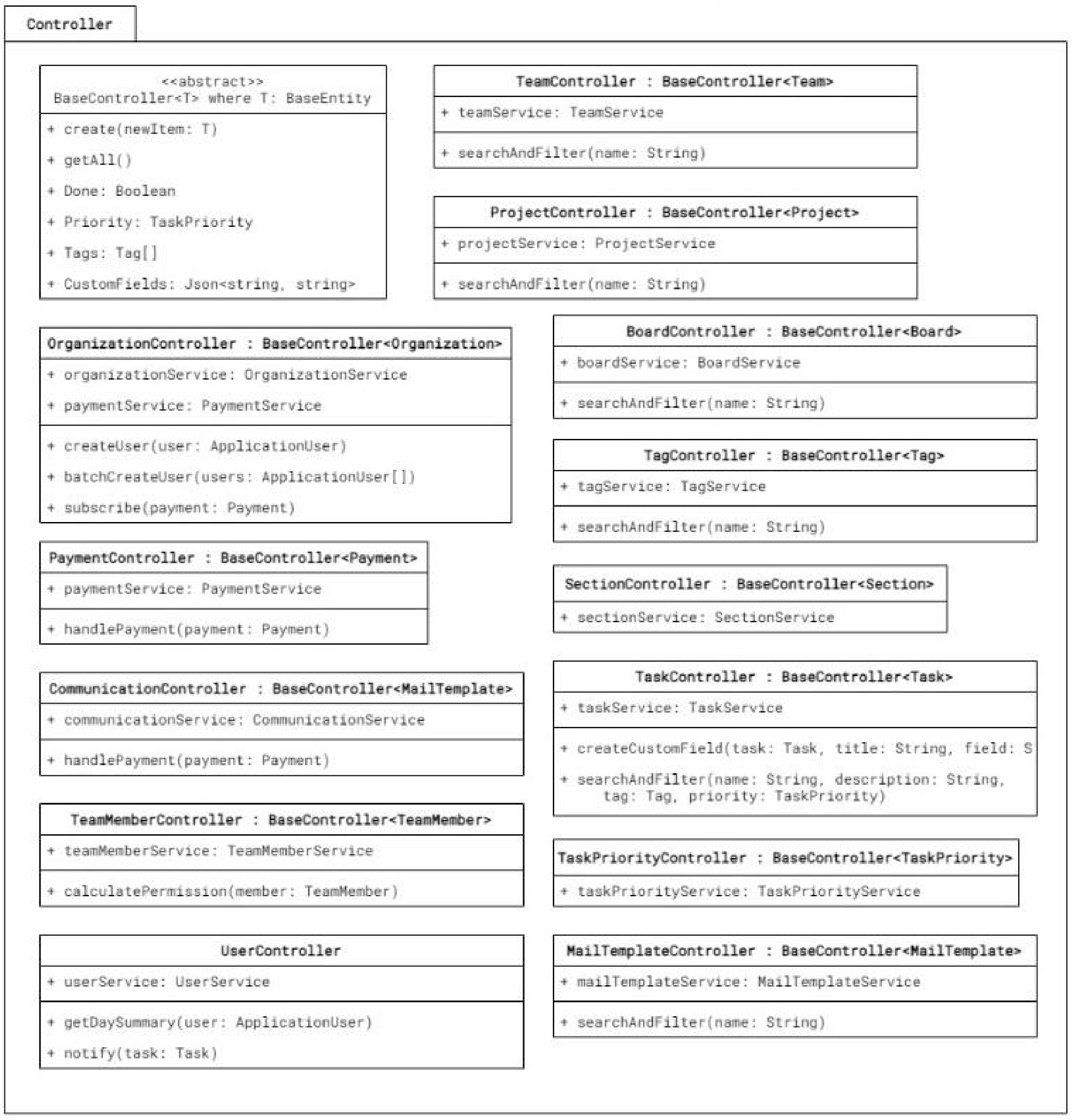
Controller
<<abstract>>
BaseController<T> where T: BaseEntity
+ create(newItem: T)
+ getAll()
+ Done: Boolean
+ Priority: TaskPriority
+ Tags: Tag[]
+ CustomFields: Json<string, string>
TeamController : BaseController<Team>
+ teamService: TeamService
+ searchAndFilter(name: String)
ProjectController : BaseController<Project>
+ projectService: ProjectService
+ searchAndFilter(name: String)
OrganizationController : BaseController<Organization>
+ organizationService: OrganizationService
+ paymentService: PaymentService
+ createUser(user: ApplicationUser)
+ batchCreateUser(users: ApplicationUser[])
+ subscribe(payment: Payment)
BoardController : BaseController<Board>
+ boardService: BoardService
+ searchAndFilter(name: String)
TagController : BaseController<Tag>
+ tagService: TagService
+ searchAndFilter(name: String)
PaymentController : BaseController<Payment>
+ paymentService: PaymentService
+ handlePayment(payment: Payment)
SectionController : BaseController<Section>
+ sectionService: SectionService
CommunicationController : BaseController<MailTemplate>
+ communicationService: CommunicationService
+ handlePayment(payment: Payment)
TaskController : BaseController<Task>
+ taskService: TaskService
+ createCustomField(task: Task, title: String, field: S
+ searchAndFilter(name: String, description: String, tag: Tag, priority: TaskPriority)
TeamMemberController : BaseController<TeamMember>
+ teamMemberService: TeamMemberService
+ calculatePermission(member: TeamMember)
TaskPriorityController : BaseController<TaskPriority>
+ taskPriorityService: TaskPriorityService
UserController
+ userService: UserService
+ getDaySummary(user: ApplicationUser)
+ notify(task: Task)
MailTemplateController : BaseController<MailTemplate>
+ mailTemplateService: MailTemplateService
+ searchAndFilter(name: String)

Figure 2: Controller Package

Controllers

This package handles requests coming to the REST endpoints and responds appropriately with the result from the services.

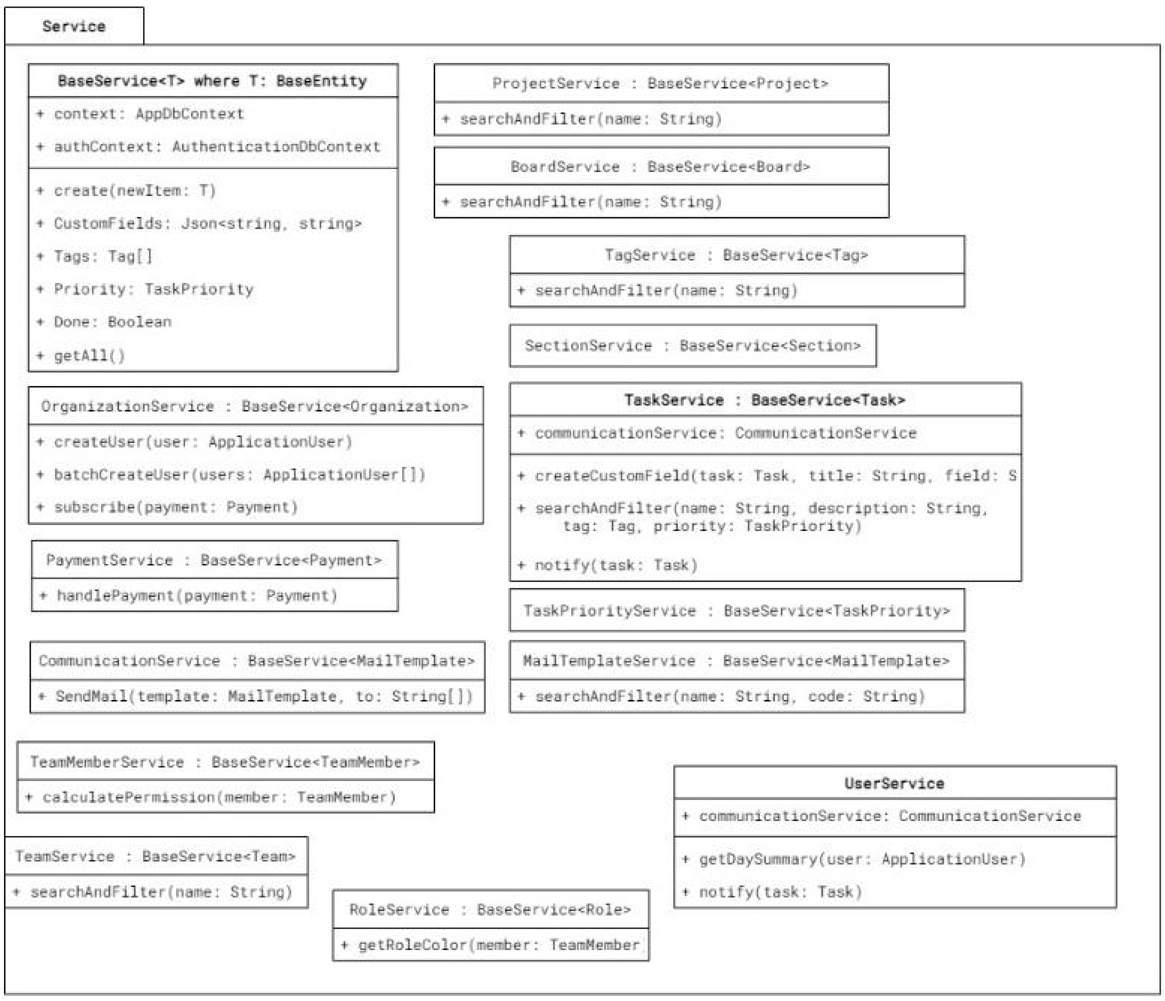
Service
BaseService<T> where T: BaseEntity
+ context: AppDbContext
+ authContext: AuthenticationDbContext
+ create(newItem: T)
+ CustomFields: Json<string, string>
+ Tags: Tag[]
+ Priority: TaskPriority
+ Done: Boolean
+ getAll()
ProjectService : BaseService<Project>
+ searchAndFilter(name: String)
BoardService : BaseService<Board>
+ searchAndFilter(name: String)
TagService : BaseService<Tag>
+ searchAndFilter(name: String)
SectionService : BaseService<Section>
OrganizationService : BaseService<Organization>
+ createUser(user: ApplicationUser)
+ batchCreateUser(users: ApplicationUser[])
+ subscribe(payment: Payment)
TaskService : BaseService<Task>
+ communicationService: CommunicationService
+ createCustomField(task: Task, title: String, field: S
+ searchAndFilter(name: String, description: String, tag: Tag, priority: TaskPriority)
+ notify(task: Task)
PaymentService : BaseService<Payment>
+ handlePayment(payment: Payment)
TaskPriorityService : BaseService<TaskPriority>
CommunicationService : BaseService<MailTemplate>
+ SendMail(template: MailTemplate, to: String[])
MailTemplateService : BaseService<MailTemplate>
+ searchAndFilter(name: String, code: String)
TeamMemberService : BaseService<TeamMember>
+ calculatePermission(member: TeamMember)
UserService
+ communicationService: CommunicationService
+ getDaySummary(user: ApplicationUser)
+ notify(task: Task)
TeamService : BaseService<Team>
+ searchAndFilter(name: String)
RoleService : BaseService<Role>
+ getRoleColor(member: TeamMember

Figure 3: Service Package

Services

This package contains the core of the business logic.

3. Class Interfaces

3.1 UI Interfaces

Class	ProfileUI
Displays the profile of the user	
Attributes	
name	String-name of the user
email	String-name of the email
image	Image-image of the email
notes	List-notes of the user
projects	List-projects of the user
Methods	
editName()	edits the name
changeImage()	changes the image
showProjects()	displays project details

Table 2: ProfileUI Class Interface

Class	TeamsUI
Display of the project teams	
Attributes	
nameList	List-name of the team members
lastnameList	List-last name of the team members
projectList	List-name of the projects

projectIds	List-ids of the projects
Methods	
editTeams()	edits the team items
deleteMember()	deletes a team member
addMember(String name, String lastName, String projectName, int projectId)	adds a team member

Table 3: TeamsUI Class Interface

Class	ProjectsUI
Displays the projects and their details	
Attributes	
projectName	String-name of the project
projectId	int-id of the project
projectDetail	String-detail text of the project
tasks	String-tasks of the user
finishedTasks	String-finished tasks of the user
Methods	
checkTask()	checks the task as finished

Table 4: ProjectsUI Class Interface

Class	CalendarUI
Displays the calendar of the user	
Attributes	
title	String-title of the event

repeat	boolean-shows if the event is repetetive
location	String-the location of the event
startDate	String-start date of the event
endDate	String-end date of the event
description	String-description of the event
Methods	
addTitle(String title)	adds event to the calendar
saveEvent(String location, String endDate, String startDate, String description, boolean repeat)	saves the details of the event

Table 5: CalendarUI Class Interface

Class	SettingsUI
Displays the settings of the application	
Attributes	
getHelp	Button-displays the help box
feedback	String-feedback of the user
notifications	List-notification preferences of the user
Methods	
saveFeedback(String feedback)	sends the feedback of the user
savePreferenceButton(Button button)	saves the notification preferences of the user

Table 6: SettingsUI Class Interface

Class	TasksUI
Displays the tasks of the user	
Attributes	
taskList	List-tasks of the user
taskTitle	String-title of the task
durationTask	int-duration of the task
progressTask	Bar-progress bar of the task
Methods	
addTaskButton(Button button)	adds a new task

Table 7: TasksUI Class Interface

Class	HomeUI
Displays the home page of the user	
Attributes	
calendarBox	Box-shows the calendar
ganttChart	Chart-shows the gantt chart
noteBox	Box-shows the notes of the user
notes	String-notes of the user
Methods	
saveGanttButton(Button button)	saves the details of the Gantt chart
saveNotesButton(Button button)	saves the notes

Table 8: TasksUI Class Interface

Class	SignUpUI
Sign up page of the application	
Attributes	
name	String-name of the user
lastName	String-last name of the user
userName	String-user name of the user
email	String-email of the user
password	String-password of the user
passwordConfirm	String-confirmation of the password
Methods	
signUpButton(Button button)	saves the details of the new user

Table 9: SignUpUI Class Interface

Class	SignInUI
Sign in page of the application	
Attributes	
userName	String-user name of the user
email	String-email of the user
password	String-password of the user
Methods	
signInButton(Button button)	signs in the user

Table 10: SignInUI Class Interface

3.2 Back-end and Server Interfaces

Class	BaseController<<abstract>>
The main controller that accesses other controllers.	
Attributes	
Done	Boolean
Priority	TaskPriority
CustomFields	Json<string, string>
Methods	
create	Creates new controllers/items.
getAll	

Table 11: BaseController Class Abstract

Class	OrganizationController
The main controller of the organization.	
Attributes	
organizationService	OrganizationService
paymentService	PaymentService
Methods	
createUser	Creates new users for the organization.

batchCreateUser	Creates a group of users for the organization.
subscribe	The payment information of the organization.

Table 12: OrganizationController Class

Class	UserController
User controller.	
Attributes	
userService	UserService
Methods	
getDaySummary	Give information to the user about the day.
notify	Notifciation method for tasks

Table 13: UserController Class

Class	TaskController
The task controller for each task.	
Attributes	
taskService	TaskService
Methods	

createCustomField	Create custom filter options for tasks while searching.
searchAndFilter	Filter the tasks according to given information

Table 14: TaskController Class

Class	TeamMemberController
The members of the team are defined here.	
Attributes	
teamMemberService	TeamMemberService
Methods	
calculatePermission	According to permissions, members could affect other members.

Table 15: TeamMemberController Class

Class	CacheRepository
The cache that is going to be written into the database.	
Attributes	
redisClient	
Methods	
Set<T>	Set the information attributes like key, value, timeout.

Get<T>	Get the info according to the key string.

Table 16: CacheRepository Class

Class	AppDbContext
The class, holding the information of organizations.	
Attributes	
Organization	DbSet<Organization>
TeamMembers	DbSet< TeamMember>
Roles	DbSet< Role>
Boards	DbSet< Board >
Projects	DbSet< Project >

Table 17: AppDBContext Class

4. References

[1] React. “ *React*”. [Online]. Available: https://tr.reactjs.org/. [Accessed: 24 Dec-2021].

[2] Next js. “ *Next js*”. [Online]. Available: https://nextjs.org/. [Accessed: 24 Dec-2021].

[3]PostgreSQL. “ *PostgreSQL*”. [Online]. Available: https://www.postgresql.org/. [Accessed: 24 Dec-2021].

[4] ”Unified Modeling Language”. [Online]. http://www.uml.org/. [Accessed: 4- Feb-2021]

[5] IEEE REFERENCE GUIDE. 2020. [Online]. https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. [Accessed: 4- Feb-2021].

[6] “Unified modeling language.” https://www.visual-paradigm.com/guide/ uml-unified-modeling-language/what-is-uml/. [Accessed: 5- Feb- 2021]