Bilkent University Department of Computer Engineering

Senior Design Project

Project Name: Cronus

High Level Design Report

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

Supervisor: İbrahim Körpeoğlu

Jury Members: Shervin Arashloo and Hamdi Dibeklioglu

High Level Design Report Dec 24, 2021

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

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 Purpose of the system

Cronus is a project management tool, where members of a project can be assigned tasks by a project leader and everyone can follow the repetitive tasks in the project. The application is planned to work as a web application. The tasks will be created and they will be assigned to the members. Therefore, the assignment of the tasks to the team mates will be playing an important role. The system will keep the created issues and control if the individuals complete their tasks. Especially for the repetitive tasks it is hard to control the tasks but our application will handle the repetitive tasks too.

1.2 Design goals

In our design, it is significant to consider simplicity and a user-friendly user interface. The reason behind the simplicity is to provide a wide range of expected users. Because of this, the application must be easy for the users to use and understand. There are some rules that should be explained according to our application.

1.2.1 User Related

- Users should be able to sign up to create their profiles
- Users should be able to login/logout through the application
- Users may hide their personal information which is not required.
- A user should be able to create a team and add team members to the team they created.
- Users should be able to create a project for a team.
- Users should be able to create tasks and assign the task to a specific user.
- Users should be able to download the web application in order to open and use it.
- After a task is completed by a specific user, other members in the same project should see the change.

- The responsible user should be able to add a new team member or delete that member from the group.
- For the tasks, the user can add new tasks or delete tasks according to the schedule.

1.2.2 System Related

- A login and register system for the application database should exist for any possible fraud action.
- The system should have a list of members that are registered.
- The system should have a list of issues that is created by a user that is to be assigned.
- The system should be available in the web environment for free.
- The system should be available to show the completed tasks of a user in a team.
- The system should be able to maintain and view the reviews acquired by a user.

1.2.3 Usability & Accessibility Related

- The application should be available on the web environment, since the users in a team should reach to the application easily in order to contact in an efficient way.
- The application should have a neat user interface that should provide an easy usage to the users for the repetitive tasks.
- For repetitive tasks, the application should provide a feasible usability experience.
- The interface should represent the tasks in a neat and bigger visual view that the most important part for the users to see is for tasks and their completion.
- The application should notify the user for specific cases as the coming deadlines or for the tasks that are completed or changes. There should

be an option to choose whether to get a notification from the application or not.

1.2.4 Reliability Related

- The application should not mix the tasks assigned to the team members. Tests for this requirement should be made.
- The system should have an error handling system for the cases that an error may occur and the handler should notify the user for that error.
- The server of the application should be capable of working for the case of maximum numbers using the application.
- The maximum number of users should be fixed to 1000 at first, this number can be increased if the user number is about to be more in time.

1.2.5 Privacy & Security Related

- The server must be secure enough against adversaries.
- The application should not reach the needless data from the users web browser that should be kept private and protected from any possible abuse.

1.2.6 Efficiency Related

- The application should work properly in older versions of operating systems of computers.
- The application’s total storage size should not take more than 200 MB. With this way, it becomes easier to store and access.
- The application should detect the projects that were finished a long time ago. They should be deleted from the system for the purpose of storage efficiency.
- The application should detect the users who are not using the application for a long time. those accounts should be deleted within the

user's knowledge. The notification to the user can be done by sending an email to her email address. When the decision is to delete the account, all of the information about that person should be deleted from the system.

1.2.7 Extensibility Related

- According to the upgrades and improvements, there could be changes. The object documentation of the application should be open to the change.
- The source codes of the application should be commented and explained detailly for the new attenders of the software team.

1.2.8 Maintainability Related

- The application should be separated into modules in a clear way that one change of a specific part should not affect the other parts. For that, the architecture and design is significant.
- The server should be implemented well enough for the possible change of maximum users of the application.

1.2.9 Legality Related

- The private information of the users should not be shared with third parties.
- During the registration, the application should notify that the application does not take any responsibility for any illegal event performed by the users and on any illegal events' consequences on any user.

1.2.10 Performance Related

- The starting process of the application should not take more than 10 seconds.

- the creation of a task and assignment of it should not take more than 10 seconds. ıf it takes more than 10 seconds, the user should be notified with a reason.

1.3 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.
Issue: Issue is named for the tasks in the project.
Client: A user of the application. So, the client side of Cronus is the user interface side that a user can manage tasks or complete the given tasks.
Server: The back-end of the application. The information about the projects, teams and tasks are kept here.

1.4 Overview

Cronus is a web application that is designed for the project management process and for people who want to assign tasks and follow them. To be able to reach more users, the application will be free to use.

To make the reader have a better understanding of the application, it is thought by the team that it might be useful to explain some of the most important requirements in plain English sentences also, right along with upcoming sections. The application requires an account for the user to attend to the system.

After creating an account, the type of the user is determined, which can be the role manager, a participant in the team or admin. According to the type of the user,

there are different actions permitted. The manager of the project opens a new project and adds members to the team of that project. Roles are given to each member which can be creating, deleting, editing issues; managing roles or managing the project. The role manager assigns roles and issues to the team members. With the help of the application, everyone can see who has accomplished their task.

As a significant difference from other project management tools, our application helps to maintain the repetitive task management in the project. The user with the permission can change the role of the member, assign a new one or create a new assignment and determine the properties of the new role. The properties of the issue are due date, issue type, requirements or the period of time needed for it. Any issue may be deleted or renewed according to the plan by the manager. Every task completed is checked by who is assigned and it is seen by the other members in the team.

All the details on the application's requirements will be discussed in the next 4 sections. Later on, system models will be shown as visuals.

2. Proposed Software Architecture

2.1 Overview

Designing the software architecture and examining subsystem decomposition for Cronus will allow deeper understanding for the project structure as well as highlighting any connections between different subsystems. The following sections will develop the structure of the software architecture and proposed technologies for *Cronus*.

2.2 Subsystem decomposition

The system is composed of two main models — client and server. The client

architecture will be implemented as a web application using Next.JS/React, and the server architecture will be a *HTTP* server using a *REST API* with a *WebSocket* and *Webhook* server for application's pub/sub needs developed with *.NET* for the server itself, *PostgreSQL* database for the persistent data management and *Redis* for any necessary caching operations.

Client-side application will consist of loosely coupled components collected into a usable format in different pages, together with a store for local data storage during application runtime. Each component will handle the definition and the styling of the user interface, react to any user events such as clicks, scrolls and keyboard inputs. Components will also handle connecting to the server-side application for necessary data and send requests for any changes made by the user.

Server-side application will use a *CQRS* architecture with a mediator pattern for the API — the *presentation* layer —, a repository pattern implementation for both connecting to the *PostgreSQL* database for persistent data storage — the *persistence* layer — and connecting to a *Redis* server for handling caching needs — the *cache* layer. The API will handle the incoming requests from the clients and process them appropriately, then send an appropriate response following the processing of the request, using the *persistence* and cache layers as needed. The API additionally will handle authentication and authorization for the application, send events through the *Webhooks* to the subscribed clients, and handle any in-app and email notifications.

2.3 Hardware/Software Mapping

Client-side applications will be usable on any system supporting a browser capable of rendering a modern *React* application. The server-side application, the *PostgreSQL* database and the *Redis* service will be run on different *Docker* instances, using *Docker Compose.* Clients will connect to the server-side application through different protocols: *HTTPS* for interacting with the application for *CRUD* purposes, *WSS* protocol for connecting to the *WebSocket* endpoint for live updates and subscribing to the *Webhook* for getting subscribed events.

2.4 Persistent Data Management and Caching

The necessary persistent data for Cronus includes but not limited to company/management information, user information such as private personal information together with application specific information, projects created under the company, information regarding the teams and users' per team permissions, boards connected to each project, issues belonging to the boards, the completion information for each issue. Any user information will be kept until the company/management the user belongs to is deleted, barring any deletions by the admin accounts.

Aforementioned persistent data will be stored in a *PostgreSQL* database — with the exclusion of any information connected to authentication, which will be handled by *Firebase.* The private information stored will not be shared with any third parties, however anonymised information may be shared for different purposes if necessary. *PostgreSQL* was chosen over other available database systems due to being free, open source, constantly updated to include new features.

Application will also cache frequently accessed information such as permissions for users, and it will be stored in a *Redis* instance. The instance may also be used for storing short-lived and unstructured data without adding a *NoSQL* database.

Client-side applications may also store some information into *local storage* to help with load times.

2.5 Access Control and Security

To access the application, every user needs an account. An admin account can be created from the main application page, which in turn can create accounts for any members. Additionally, a user may create a personal account that will be connected to multiple admin accounts. Each account must have a valid and unique email address connected to it.

After creating a personal or an admin account the user will be sent a verification email to the provided email address. Users will need to verify their accounts before

being allowed to access the service. The verification email will be valid for an hour, before the user needs to request another verification email to be sent. If an account is unverified for more than a month, the account information will be deleted to prevent email squatting.

Accounts created by an admin account will be sent an email to the address provided during the account creation. If they click the *join* or *accept* button in the email, the account will be considered verified, and it will be added to the admin account. Then, the user will be redirected to a screen to fill in personal information and set their password.

If an account created by an admin account is requested by another admin account, the user will be asked if they want to convert their account to a personal account.

Account creation and authentication will be handled by *Firebase Authentication,* both for security purposes and due to the flexibility *Firebase* gives to the developers out of the box, such as immediate *OAuth* authentication, authentication via different services such as via *Google, Facebook, GitHub, etc.* However, any private information will be stored in the application's database and will not be shared without a thorough anonymising process. If an account is deleted, any information connected to the account will be deleted immediately as well.

2.6 Global Software Control

Requests will be initiated through user actions like creating an account, declaring an issue complete, updating a board name. Any action a user can take in the system will have either a request or a group of requests that corresponds to it. Responses from the application will depend on the request, however due to the usage of *WebSocket* and *Webhooks* a response may be sent without a corresponding request due to a change made by another user.

2.7 Boundary Conditions

2.7.1 Start-up

Users will access the application through the client web application. Due to this requirement a user wanting to access the application must have access to a web browser capable of rendering Cronus. When the web application is launched for the first time, the user will need to either create an account or login to an existing account.

During the registration a user will supply the necessary information such as email and password, full name etc. The system will then check the registration information with the existing accounts and make sure that this account doesn't already exist. The application will also verify the validity of the email address and check that password fits the requirements for passwords. If there're any errors during this process, the user will be given the error. Otherwise, the user will be redirected to the main application page and the user will be able to interact with the application.

If there's an already existing account, the user will supply the email and password for logging in. The supplied information will be used to authenticate the user. If there's no error during this process, the user will be redirected to the main application page. Otherwise, the user will be shown an error that there was an issue with the authentication.

2.7.2 Shutdown

To leave the application, the user can simply close the browser tab. If the user also wants to login from the application, they can do it through the menu before closing the tab.

2.7.3 Error

If there're any errors during the usage, the user should reload the page. They will not need to login again, as the login will be saved through a cookie unless the cookies are disabled for the page. Since everything is saved to the database, there will be no data loss due to the reload, apart from any actions left incomplete and not sent to the server before the reload.

Additionally, as a web application Cronus needs a persistent internet connection. If the device loses its connection to the internet, every interaction will fail.

3. Subsystem services

Cronus has two subsystems, one for the user and one for the server.

3.1 User

3.1.1 ClientUI

MainControl: Every view is controlled by the MainControl. Pages are displayed to the client according to the change.

ProfileUI: View of the client's own profile

TeamsUI: View of the client's teams in the different projects, members can be added, deleted or edited by the user

ProjectsUI: View of the client's different projects

CalendarUI: View of the calendar belonging to the client, events, meetings, important dates can be added to the calendar.

SettingsUI: View of the settings

TasksUI: View of the client's tasks

HomeUI: View of the client's home page

SignUpUI: View of the signing up to Cronus

SignInUI: View of the signing in to Cronus

NotificationsUI: View of the client's notifications

3.1.2 Control

ControlRequest_Client_Server: Maintains communication between the ControlRequest_Server_Client to carry information coming from the client.

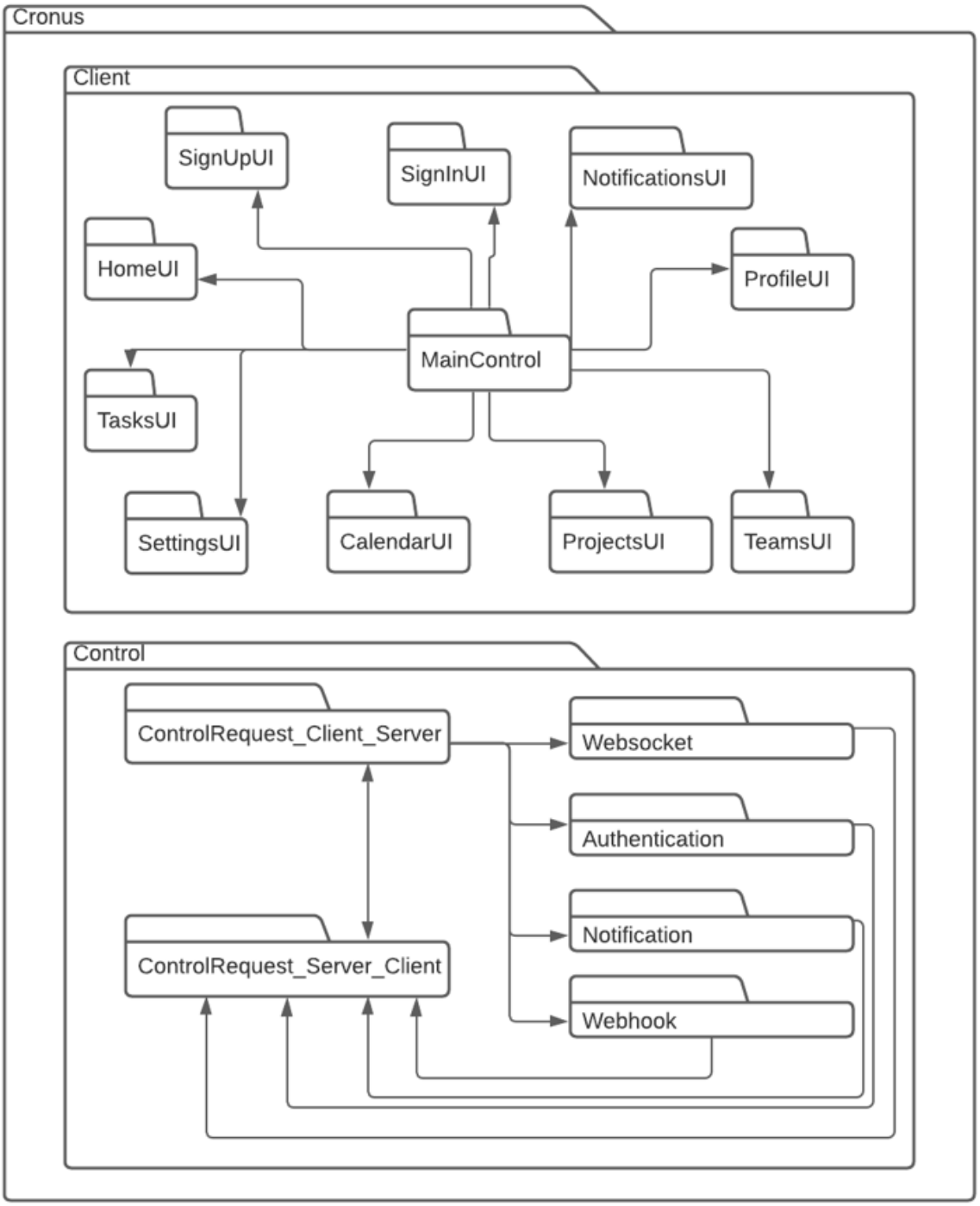
ControlRequest_Server_Client: Create responses coming from the server to the client UI.

Websocket: Sends live updates from the server to the client.

Authentication: For the protected connection with the server.

Notification: Sends any necessary notifications via in-app notification system or email.

Webhook: Publishes updates to subscribed clients based on the scopes.


Cronus
Client
SignUpUI
SignInUI
NotificationsUI
HomeUI
ProfileUI
MainControl
TasksUI
SettingsUI
CalendarUI
ProjectsUI
TeamsUI
Control
ControlRequest_Client_Server
Websocket
Authentication
Notification
ControlRequest_Server_Client
Webhook

3.2 Server

3.2.1 I/O

RestAPI: Rest API for the clients requests, carries these requests to processor for responses

3.2.2 Processor

Authentication: Authentication for the clients during server access

TaskManager: Maintains task feature to the client

AccountManager: Maintains account management for the client

BoardManager: Maintains board feature to the clients

Webhook: Publish updates to the client.

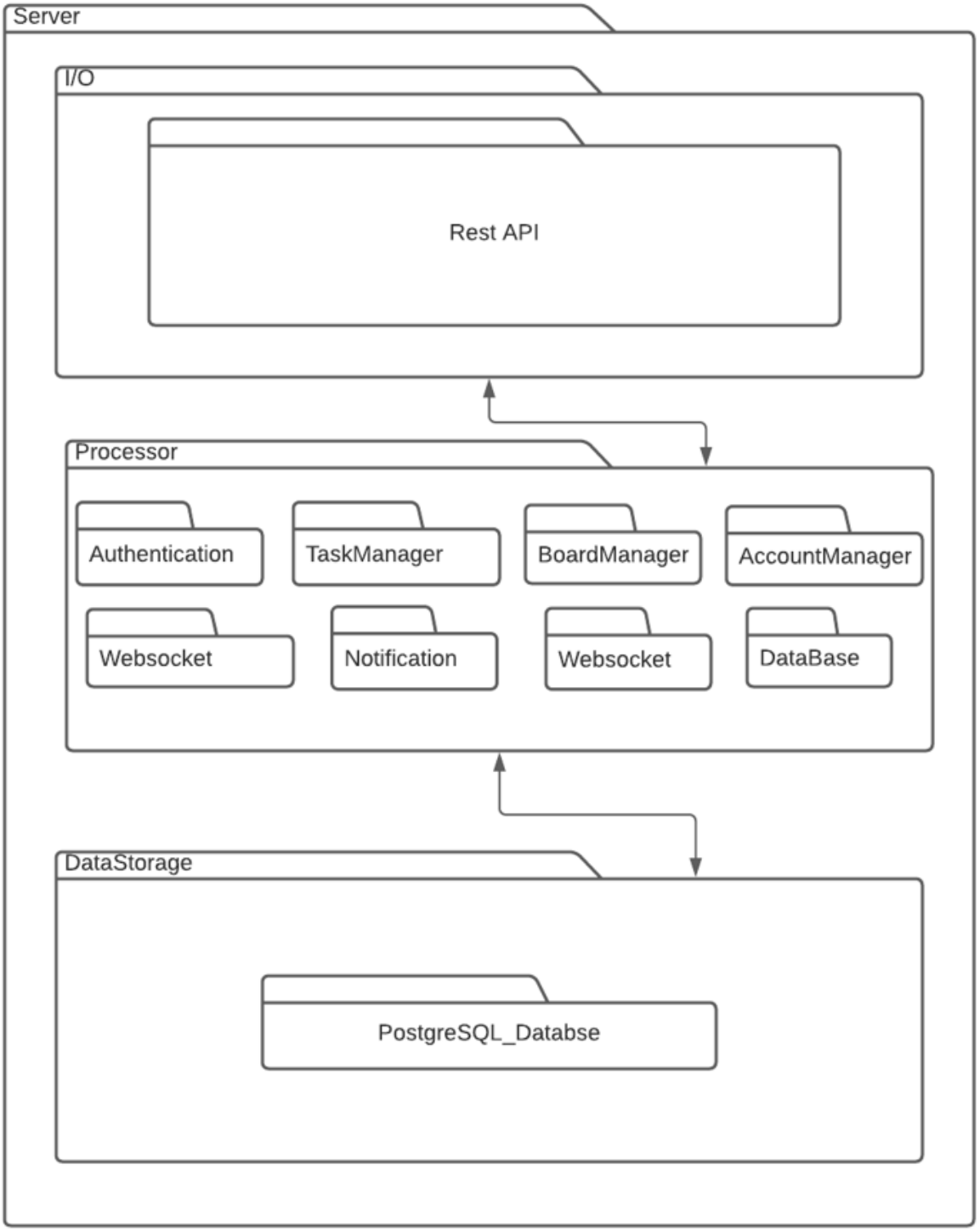
Websocket: Send live updates to the client.

Notification: Send notifications to the client.

DataBase: Maintains management of the database queries

3.2.3 Data Storage

PostgreSQL_Database: PostgreSQL database is used for data storage.



4. Consideration of Various Factors in Engineering Design

4.1 Public Health

The application can be used in every environment with the internet, so people from different places can create and manage a project together using this tool. Due to the COVID-19 pandemic[], Cronus aims for public health by minimizing the need for project team members to meet face to face.

4.2 Economic Factors

Cronus aims to increase effectiveness and reduce wasted time, so the application can help individuals or companies to profit indirectly and the employment rate may increase. Application will not require a monthly or one-time payment. Every user will be able to access the same features and everything will be free.

4.3 Environmental Factors

There are not any environmental factors to be considered.

4.4 Societal Factors

By easing up the process of project management, Cronus re-arranges many social organizations such as meetings, presentations, reports. With that Cronos reduces the chaos and conflicts that can arise between co-workers, teammates, bosses, employees, etc. As a result, an increase in social peace can be obtained.

4.5 Cultural Factors

Since Cronus will be able to be used all around the world, it aims to bring different cultures together. People from different countries will have the chance to create and manage a project together, which helps people to get to know other cultures.

4.6 Public Safety

The information about the companies and employees can only be accessed by the people who work in that company but according to the hierarchy inside the system. Through this, the application prevents undesirable accesses.

	Effect Level	Effect
Public Health	9	Physical health
Economic Factors	8	Increase in profit and employment rate
Environmental Factors	0	None
Societal Factors	7	Socially peaceful environment
Cultural Factors	5	Bringing different cultures together
Public Safety	7	Prevents third-party and unauthorized access

Table 1: Factors to consider during the design.

5. Teamwork Details

5.1 Contributing and functioning effectively on the team

In order to make everyone attend the project, we have study modules and every one is responsible for one of them as a leader. Every schedule has to be led and all group members have taken the one which they are familiar with according to the subject. In each module the leader will decide on the process and the resources needed for that module.

5.2 Helping creating a collaborative and inclusive environment

All team members should care for each other in order to create a collaborative environment. It is essential to act as a team and complement each other in weak areas to overcome difficulties. Giving a chance to every group member about an idea or perspective for the project is important to increase the performance and team positivity.

5.3 Taking lead role and sharing leadership on the team

In order to give responsibility to every member, the project is divided into pieces and each member is a leader in that part of the project. Also sharing the updates and sources was an important step for us.

6. 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]Webhook. “ *What is a webhook: How they work and how to set them up*”. [Online]. Available: https://www.getvero.com/resources/webhooks//. [Accessed: 24 Dec -2021].

[5] Rest. “ *What is a REST API?*”. [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Accessed: 24 Dec -2021].

[6] Websocket. “ *The WebSocket API (WebSockets)*”. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Accessed: 24 Dec- 20201].