**San Francisco State University**

**SW Engineering CSC 648/848**

**Milestone 2**

**Section 04 Team 05**

# Project title

Priority Hub (Code: PH)

# Team members

* Luis Aguilar (Team Lead)
* Tyler Tam (Scrum Master)
* Trevor Eichler (GitHub Master)
* Andy Li (Front Lead)
* Lam Tran (Back Lead)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 1.0 | Saturday, September 30th, 2023 | Team 05 | Draft version |
| 1.1 | Wednesday, October 11th , 2023 | Team 05 | Version to submit |
|  |  |  |  |

# Table of contents

[Project title 1](#_Toc147960527)

[Team members 1](#_Toc147960528)

[Revision History 1](#_Toc147960529)

[Table of contents 2](#_Toc147960530)

[Content and structure for Milestone 2 document for review by institutors 3](#_Toc147960531)

[1. Data Definitions 3](#_Toc147960532)

[2. Functional requirements 6](#_Toc147960533)

[1. UI Mockups and UX Flows 9](#_Toc147960534)

[2. High-level architecture 11](#_Toc147960535)

[2.1. Database organization 11](#_Toc147960536)

[2.2. Add/Delete/Search architecture 13](#_Toc147960537)

[2.3. APIs 14](#_Toc147960538)

[3. High level UML Diagrams 14](#_Toc147960539)

[3.1. High-level UML class diagrams 14](#_Toc147960540)

[3.2. High-level sequence diagrams 15](#_Toc147960541)

[4. Key risks 17](#_Toc147960542)

[4.1. Skills risks and mitigation plan 17](#_Toc147960543)

[4.2. Schedule risks 17](#_Toc147960544)

[4.3. Teamwork risks (issues related to teamwork) 18](#_Toc147960545)

[4.4. Legal/content risks 18](#_Toc147960546)

[5. Project management 19](#_Toc147960547)

# Content and structure for Milestone 2 document for review by institutors

## Data Definitions

|  |  |  |
| --- | --- | --- |
| **Primary Data Name** | **Definition** | **Usage** |
| User | Anyone who has registered for an account on the application. | Information gathered from registered users will be stored in the database and will be used later. |
| Role | The types of users on the application | This information will determine what level of access a person will have when utilizing the application (Admin level or client user level). |
| LoginLogout | Each attempt of the users whenever they sign in or signs out. | Whenever users try to login/logout, the system will store each of those attempts. |
| Task | An event that the user can customize to suit their required needs. Any type of activity that needs to be done throughout the day (Review documents, daily morning run, study math, etc.) | The list of tasks will be shown to the user once they sign into the system. |
| Category | Customize an event to specify what it relates to, such as study, work, personal, etc. | When creating a task, the user can choose which category it should belong to. |
| ActivityTracker | This table will keep track of users’ time spent on tasks that are performed throughout the day. | Users can see how they are doing throughout the day and what tasks they have done with their time spent. |
| EODSummary | An “End of day summary” will be shown to the user so that they can keep track of their performance. | The user may choose a time when their day ends and will be shown what the user has accomplished during that timeframe. |

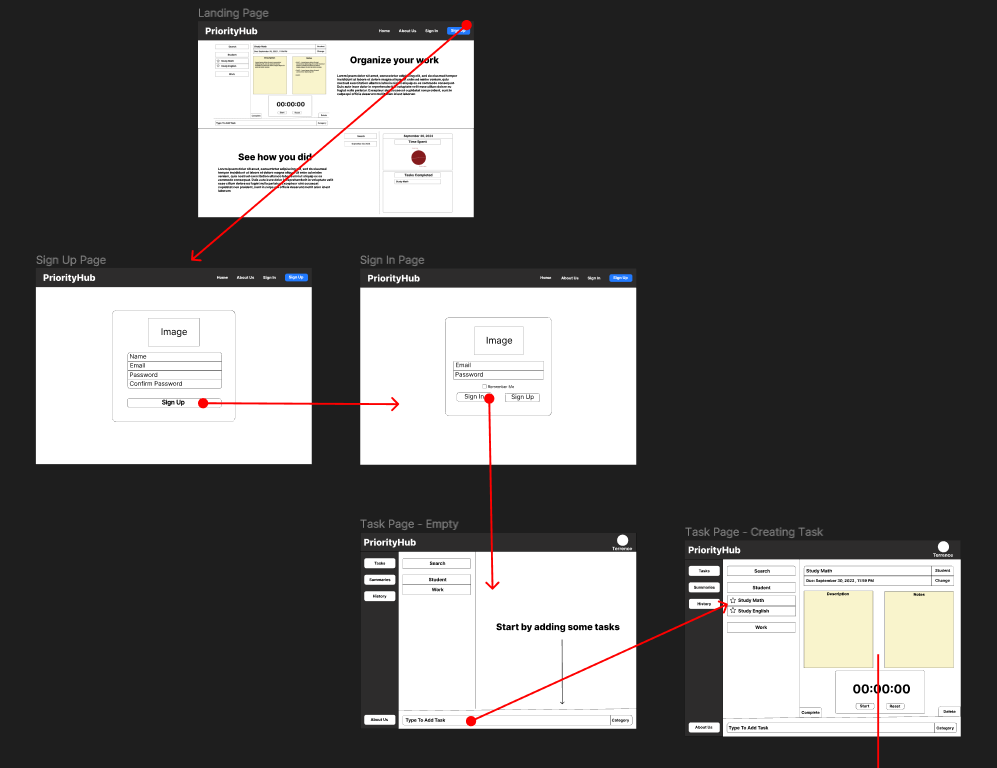
|  |  |
| --- | --- |
| **Primary Data Name** | **Sub-Data** |
| User | * ID * Username * Password * FirstName * LastName * Status * Birthday * ZipCode * Country * Address * ProfilePictureURI * EmailConfirmed * CreatedBy * CreatedDateTime * LastUpdatedBy * LastUpdatedDateTime |
| Role | * ID * Name * Description * IsAdmin * Deleted * CreatedBy * CreatedDateTime * LastUpdatedBy * LastUpdatedDateTime |
| Login/Logout | * ID * Type * UserIp * LoginDateTime * LogOutDateTime * Description * UserID * Timestamp * CreatedBy * CreatedDateTime * LastUpdatedBy * LastUpdatedDateTime |
| Task | * ID * Title * Description * CategoryID * UserID * Status * Priority * Completed * Deleted * IsFavorite * DueDate * Notes * CreatedBy * CreatedDateTime * LastUpdatedBy * LastUpdatedDateTime |
| Category | * Id * Title * Description * Deleted * Createdby * Createddatetime * Lastupdatedby * Lastupdateddatetime |
| ActivityTracker | * ID * StartTime * StopTime * Timestamp * Notes * TaskID * CreatedBy * CreatedDateTime * LastUpdatedBy * LastUpdatedDateTime |
| EODSummary | * ID * Timestamp * UserID * CreatedBy * CreatedDateTime * LastUpdatedBy * LastUpdatedDateTime |

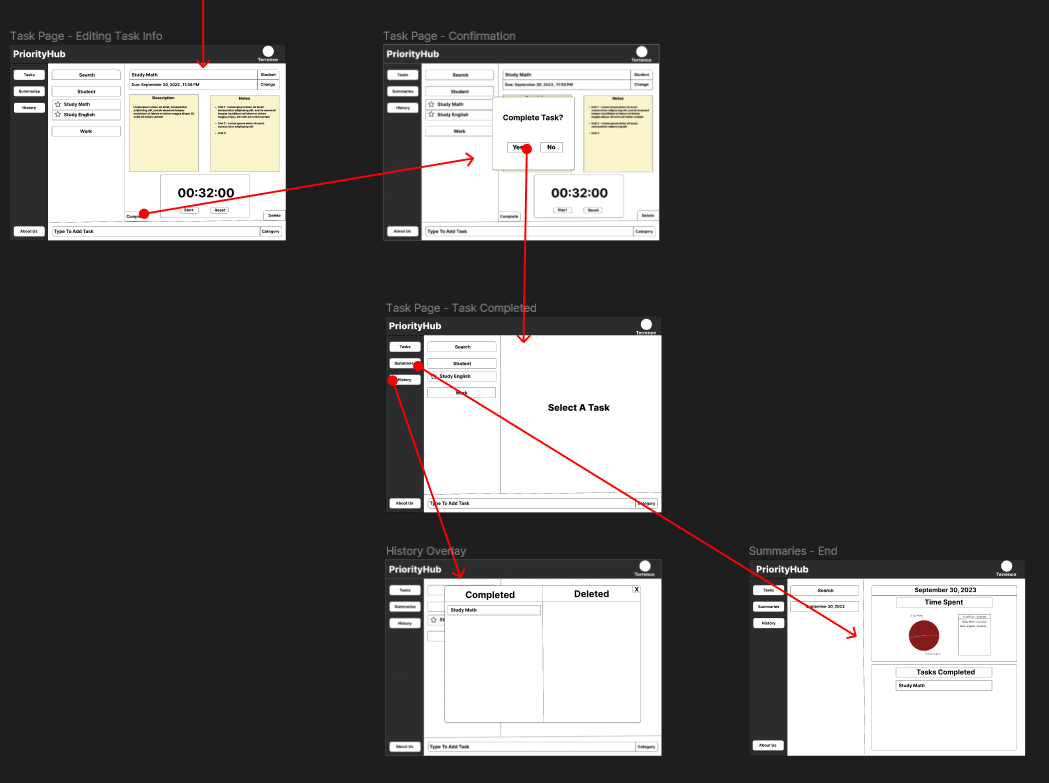
## Functional requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Must-have |  | Desired |  | Opportunistic |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ID** | **Description** | **Details** |
|  | 1 | User should be able to sign-up | * 1. User can sign up for a new account.   2. User will use their email address to create an account.   3. User will be able to create their own password |
|  | 2 | User should be able to log into the system using their email address and password. | 2.1) User can sign in with their existing account in the system.  2.2) The user should be able to press a “Sign In” button to access the system. |
|  | 3 | The system should be able to show an error message box when the user is not able to access its account | 3.1) A modal should appear with the error message.  3.2) The modal should have an “OK” button to let the user close it. |
|  | 4 | The system should be able to validate that the email address provided by user is syntax valid. (Must have @, a domain, etc.) | 4.1) The email should comply with the regular emails format (no special characters, @, has a domain, etc.) |
|  | 5 | The system should be able to validate that the credentials provided by the user exists in the application’s database. | 5.1) The system will compare the username and password provided by the user against what’s in the database.  5.2) The system will show a success/error message if attempt was successful or not. |
|  | 6 | User should be able to reset their password. | 6.1) User can reset or update their password  6.2) |
|  | 7 | The system should be able to show a confirmation/error message box when the password was/wasn’t reset successfully. | 7.1) The system will show a success message box if the user’s password was reset successfully  7.2) The system will show an error message box if the user’s password wasn’t reset successfully. |
|  | 8 | User should be able to set its profile (name, birthday, etc.) | 8.1) Once user creates their account, it will have an option to update their profile.  8.2) The option will be in a dropdown menu in the icon button in the top right of the application.  8.3) The user will be able to update their name, birthday, address, zip code, etc. |
|  | 9 | User should be able to create a task including name, due date, priority, category, and notes. | 9.1) User will be able to create a Task by typing its title.  9.2) System will create the Task with default values, in case the user does not fill all of them |
|  | 10 | The system should be able to show a confirmation/error message box when a task is/isn’t successfully created. | 10.1) The system will show a confirmation message box if the task was created successfully.  10.2) The system will show an error message box is there was an issue when creating the task. |
|  | 11 | User should be able to edit a task. | 11.1) User will be able to select a Task and edit their fields. |
|  | 12 | The system should be able to show a confirmation/error message box when a task is/isn’t successfully edited. | 12.1) The system will show a confirmation message box if the task was edited successfully.  12.2) The system will show an error message box if the task’s updates were not saved in the database. |
|  | 13 | User should be able to delete a task | 13.1) The user will be able to press a “Delete” button to remove a task.  13.2) The system will perform a “soft delete” in the database. This means that in the “Deleted” column (Boolean), the value will be updated to True. |
|  | 14 | The system should be able to show a confirmation/error message box when a task is/isn’t successfully deleted. | 14.1) The system will show a confirmation message box if the task was deleted successfully.  14.2) The system will show an error message box if the task was deleted successfully. |
|  | 15 | User should be able to see all tasks created in a list. | 15.1) The user will see their tasks in a list grouped by their category. |
|  | 16 | User should be able to browse the tasks by category. | 16.1) The user will be able to search through their tasks using a search bar |
|  | 17 | User should be able to search a task by its name or code. | 17.1) The user will be able to use the search bar at the top of their task’s list to filter tasks by name or code. |
|  | 18 | User should be able to keep track of the time spent in a task. | 18.1) User will have a stopwatch functionality within each task, in which they will be able to keep track of the time multiple times.  As an example, the user may track the time of the task “study math” for 1 hour in the morning and 2 hours in the evening, therefore, within that “study math” task, the user has 2 records in their stopwatch. One from the morning, and the other from the evening. |
|  | 19 | User should be able to start/stop a timer when keeping track of a specific task. | 19.1) Within each task, the stopwatch will have the start and stop buttons. Every time the user starts and then stops the time, it will create a new stopwatch record for that period.  19.2) The stopwatch records are created within each Task. So, each stopwatch record belongs to a specific Task. And a Task can have multiple stopwatch records. |
|  | 20 | User should be able to check all time spent on specific tasks through multiple charts. | 20.1) The system will be able to show the user the amount of time spends through a series of charts (bar charts, pie charts).  20.2) The system will group this amount of time by the categories, days, and other dimensions. |
|  | 21 | The system should be able to send/display the user an “End of the day” (EOD) summary including the tasks and time they spent. | 21.1) The system will show an “End of the day” option to let the user see their time spent along specific days.  21.2) The user will be able to see the amount of time they spent, the number of tasks they completed, etc. |

## UI Mockups and UX Flows





## High-level architecture

### Database organization

**Users**

|  |
| --- |
| **Role** |
| ID int PK |
| Name nvarchar(50) |
| Description nvarchar(255) |
| IsAdmin tinyint(1) |
| Deleted tinyint(1) |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

|  |
| --- |
| **User** |
| ID int PK |
| Username nvarchar(100) |
| Password nvarchar(255) |
| FirstName nvarchar(100) |
| LastName nvarchar(100) |
| Status nvarchar(20) |
| Birthday date |
| ZipCode nvarchar(20) |
| Country nvarchar(50) |
| Address nvarchar(255) |
| ProfilePictureURI nvarchar(255) |
| EmailConfirmed tinyint(1) |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

|  |
| --- |
| **UserRole** |
| ID int PK |
| UserID FK -> User.ID |
| RoleID FK -> Role.ID |

|  |
| --- |
| **LoginLogout** |
| ID int PK |
| Type nvarchar(50) |
| UserIp nvarchar(50) |
| LoginDateTime datetime |
| LogOutDateTime datetime |
| Description nvarchar(255) |
| UserID FK -> User.ID |
| Timestamp datetime |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

**Tasks**

|  |
| --- |
| **Task** |
| ID int PK |
| Title nvarchar(50) |
| Description nvarchar(255) |
| CategoryID FK -> Category.ID |
| UserID FK -> User.ID |
| Status nvarchar(20) |
| Priority nvarchar(20) |
| Completed tinyint(1) |
| Deleted tinyint(1) |
| IsFavorite tinyint(1) |
| DueDate datetime |
| Notes nvarchar(1000) |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

|  |
| --- |
| **Category** |
| ID int PK |
| Title nvarchar(50) |
| Description nvarchar(255) |
| Deleted tinyint(1) |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

**Activity Tracker**

|  |
| --- |
| **ActivityTracker** |
| ID int PK |
| StartTime time |
| StopTime time |
| Timestamp datetime |
| Notes nvarchar(1000) |
| TaskID FK -> Task.ID |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

**“End of the Day” Summary**

|  |
| --- |
| **EODSummary** |
| ID int PK |
| Timestamp datetime |
| UserID FK -> User.ID |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

|  |
| --- |
| **EODNote** |
| ID int PK |
| Notes nvarchar(1000) |
| Deleted tinyint(1) |
| EODSummaryID FK -> EODSummary.ID |
| CreatedBy nvarchar(100) |
| CreatedDateTime datetime |
| LastUpdatedBy nvarchar(100) |
| LastUpdatedDateTime datetime |

### Add/Delete/Search architecture

|  |  |
| --- | --- |
| **Add/Delete/Search architecture** | **Functional Requirement** |
| Add/Delete/Search for Users | When users register |
| Add/Delete/Search for UserRole | When users registers and a role is assigned. |
| Search/Display for Role | When system tries to find user’s role |
| Add/Search for LoginLogout | When users sign in or sign out |
| Add/Edit/Delete/Search/Display for Task | When users manage their tasks |
| Search for Categories for Tasks | When users filter the tasks by category |
| Add/Display for ActivityTracker | When users record the time spent on a task (stopwatch). |
| Add/Search/Display EODSummary | When users go to “End of the day summary” option and filter the days by the date they are looking for. |
| Add/Display EODNotes | When users want to add notes to a specific day in the “End of the day summary” page. |

**Technical feasibility of DB operations**

We took the functional requirements into account when we defined the DB operations.

For example, when users want to register himself/herself into the system, the system will create a record in the **User** table, look in the **Role** table for a specific ***RoleID*** to create a new record in the **UserRole** table, this way the users have a role assigned.

As another example, if the user wants to filter the tasks by category, the system will look in the **Category** table, get the relevant **CategoryID**, and extract the tasks information from the **Task** table. This way, searching for tasks by category is viable.

Finally, each table has its own primary key, while others have their foreign key as well.

The primary key will help keep the data consistent and avoid information loss scenarios.

Also, this way we can exactly identify each record in each table.

### APIs

Our API is created using the Express web application framework.

It will allow for all the Add/Edit/Delete/Search/Display methods mentioned above.

We are using fetch to perform the calls to our backend.

For our new libraries in the backend, we are using the following:

* express
  + For REST API project structure
* mysql2
  + For connection with out MySQL database
* dotenv
  + To enable the use of environment variables
* cors
  + To enable CORS in our middleware and allow incoming requests.

## High level UML Diagrams

### High-level UML class diagrams

A diagram of a company's flowchart

Description automatically generated

### High-level sequence diagrams

User signs up

A diagram of a person's account

Description automatically generated

User signs in

A diagram of a work flow

Description automatically generated

## Key risks

### Skills risks and mitigation plan

* Future milestones could be postponed if research into a particular tool, technique, or idea takes longer than anticipated.
  + Mitigation Plan:
    - Begin early, no procrastination.
    - Search through many resources before beginning/proper research.
    - Discuss the study materials in a group conversation, don’t leave anyone hanging on information they should know.
* Any team member's study plan may be impacted by other variables, such as a family emergency or burden from other classes, which may delay the timely deployment of a particular feature.
  + Mitigation Plan:
    - Begin early to avoid entirely/severity of these factors of life.
    - Avoid putting things off, again, avoid procrastinating, stay on top of things.
    - Think ahead. If a team member has something they need and knows they have something they need to attend in the near future, get it done early.
* A team member may become frustrated, anxious, and put off the next work if they can't fix a serious error in the code for days.
  + Mitigation Plan:
    - When following instructions or tutorials, don't skip steps. This can lead to issues resulting in frustration.
    - Ensure team members comprehend the existing code well before moving on to the next stage or building upon the current code.
    - If you don't grasp a certain section of the code, ask in team Discord, communicate, this can also help other learn.

### Schedule risks

* Integrating codes can get chaotic if a team member needs to adjust and change plans while working on a specific area of the project but doesn't inform other team members who are preoccupied with their task.
  + Mitigation Plan:
    - Ensure that everyone is working toward the same objective.
    - Be proactive to prevent problems caused by improper or nonexistent communication.
* The following milestone will be postponed if the team lead does not clearly communicate to the team the deliverables for the upcoming milestone.
  + Mitigation Plan:
    - Before establishing the deliverables, consider the feedback from previous milestones, and the instructors.
    - Find out from the team members what they are most excited about working on.
    - Understand group members priorities, other work, and schedules well.
    - Create dynamic plans so that they are able to be changed as needed.

### Teamwork risks (issues related to teamwork)

* Other tasks are delayed if two team members end up working on the same project functionality because of poor communication.
  + Mitigation Plan:
    - Focus on having good communication with all other group members.
    - Ensure that everyone is aware of who is working on what task at each stage, use the JIRA board the way that it is supposed to be used.
* A team member who is responsible for building functionality on top of another team member's code will have a difficult time if the former team member doesn't carefully manage the code structures and files.
  + Mitigation Plan:
    - Engage in having a weekly code review. Maybe during team meetings.
    - When a file starts to become complex, start debugging and modularizing the code before it becomes to late.
    - Learn and adhere to the team's programming language's style documentations. Keeping things consistent and neat.
* If a member of the front-end team and a member of the back-end team do not cooperate in assisting or educating one another, the functions will not be complete and/or will not have the desired qualities.
  + Mitigation Plan:
    - Hold weekly meetings for the entire team
    - Hold each team member accountable for using the Jira board.
      * Unspoken communication.
    - Actively ask one another questions about things one is curious or confused about.

### Legal/content risks

* If a team member attempts to change a library function just to make his feature works, the team would be infringing the MIT license of the library in case we decided to commercialize the application.
  + Mitigation Plan:
    - Have a meeting with the entire team to set the rule that no library should be edited, it can only be used as-is.
* If we were to decide as a team to deploy our app onto the internet, we would need to comply with data privacy laws, such as the California Consumer Privacy Act (CCPA) in the United States, when handling user data.
  + Mitigation Plan:
    - We must be aware of the California Consumer Privacy Act as a team.
    - If we were to deploy our web app, we would need to read over this act and make sure that we comply with it.

## Project management

As a team, we had 2 front-end developers, 2 back-end developers and a full-stack developer (team lead).

The full-stack developer developed a minimum viable structure (initial codebase with a defined folder structure), so the team can start developing both frontend and backend.

The front-end developers collaborated to develop a first version of the theme for the application and the UX Flows using Figma. While one developer was doing the Figma wireframes, the other developer started implementing the UI for the vertical prototype.

Meanwhile, the backend developers started with the DB organization, UML diagrams, and map all types of requests (GET/POST) needed for the vertical prototype. At the same time, the full-stack developer configured the web server and performed the connectivity tests.

We had our weekly meetings on Fridays besides the in-class meetings over the past 2 weeks and discuss about the current status of our assigned tasks for the completion of the Milestone 2. We utilized those Monday meetings to [1] configure the local environment of each team member, [2] troubleshoot any technical issue any team member had, [3] brainstorm ideas to define the UX of our application, and [4] check overall progress of each deliverable.

As a team lead, I found that mapping all the required work, starting early and follow up with each member in case of questions or doubts were very critical to complete the 2 deliverables for the Milestone.

For the next milestones, the team has agreed to keep using a project management tool (JIRA) to organize ourselves. Also, we saw the benefit of using Confluence (collaboration tool) to map all the important information about our infrastructure (the list of libraries we are using, which is the link to our shared folder in OneDrive, which is the version of MySQL we are using, which is our website URL, etc.). With Confluence, all the information is centralized in one place, so most questions about infrastructure details are answered by just looking at the pages we created.

Finally, we will keep using Discord as the main communication tool. This will be used for online meetings and chat communications.