SWE 6623

Summer 2024

Megan Dollar

KSU ID 000232814

FINAL REPORT

**ASSIGNMENT 1**

Background

* I am a big fan of using to do lists. I write one every day and have them hanging around on pieces of paper cluttering up my desk. It is not only a disorganized approach to task management, but should the paper go missing, so too does my agenda for the day. A simple task management application could help users like me to organize tasks, prioritize tasks and keep track of each task’s status throughout the day. A clear and easy to user interface will allow users to jump right in and start using the app with no learning curve. Adding a calendar view will allow users to select a day and plan ahead with tasks or check in on their tasks for each day. Users will be able to add, edit, view, and delete tasks in order to handle their daily activities more efficiently.

Programming languages and tools

* PHP for server side scripting and back-end logic
* MySQL for database management and storing task data
* Javascript, HTML, CSS for the web applications front-end
* VS Code IDE
* phpMyAdmin to manage the MySQL database
* XAMPP to set up a local Apache server and MySQL db
* Bootstrap to create responsive and cross platform UI
* FullCalendar JS library to implement calendar functionality
* BrowserStack for cross-platform testing

Timeline

Week 1:

* Outline features and create simple prototypes of the app which will help define the scope.
* Install tools and libraries listed above and setup development environment.

Week 2:

* Create forms for adding, editing and deleting tasks and implement the front end using the prototypes.

Week 3:

* Set up PHP in XAMPP and create a DB schema for storing tasks in MySQL. Use PHP and connect front and back end to allow for task creation, reading, updating, and deleting functionality.

Week 4:

* Use FullCalendar to create a calendar view which will allow users to display tasks by date.
* Develop capabilities for users to add tasks to specific dates and display tasks for the selected dates.

Week 5:

* Improve the UI by adding filtering and sorting of tasks.
* Use BrowserStack to test that the app is responsive and user-friendly on all devices

Week 6:

* Finalize any documentation needed for the app
* Release the app

Future Development\*\*

* Transform the web app into a mobile app available on the Apple App store or Google Play store
  + Use React.js
  + Set up developer accounts with apple and google play
  + Submit to each store\*\* many steps here
* Create iCalendar file with the task data to allow users to share the tasks with their Google Calendar or Apple Calendar.

**ASSIGNMENT 2**

**Phase 1: Analysis**

Step 1.1: Gather Requirements

* Activity 1.1.1 Analyze existing task management applications
* Activity 1.1.2 Make a list of potential stakeholders and their requirements
* Activity 1.1.3 Document requirements
* Activity 1.1.4 Finalize requirements

**Phase 2: Design**

Step 2.1 Design UI

* Activity 2.1.1 Create wireframes
* Activity 2.1.2 Create UI mockups
* Activity 2.2.3 Finalize UI design

Step 2.2 Design DB

* Activity 2.2.1 Define DB schema
* Activity 2.2.2 Set up DB in XAMPP
* Activity 2.2.3 Review and finalize DB design
* Activity 2.2.4 Populate DB with test data

**Phase 3: Development**

Step 3.1 Frontend Development

* Activity 3.1.1 Create HTML markup
* Activity 3.1.2 Style with Bootstrap and CSS
* Activity 3.1.3 Integrate jQuery as needed
* Activity 3.1.4 Implement FullCalendar library

Step 3.2 Backend Development

* Activity 3.2.1 Set up PHP
* Activity 3.2.2 Write PHP for CRUD operations
* Activity 3.2.3 Add user authentication
* Activity 3.2.4 Integrate with DB

Step 3.3 Integration

* Activity 3.3.1 Integrate frontend with backend
* Activity 3.3.2 Test integration

**Phase 4: Testing**

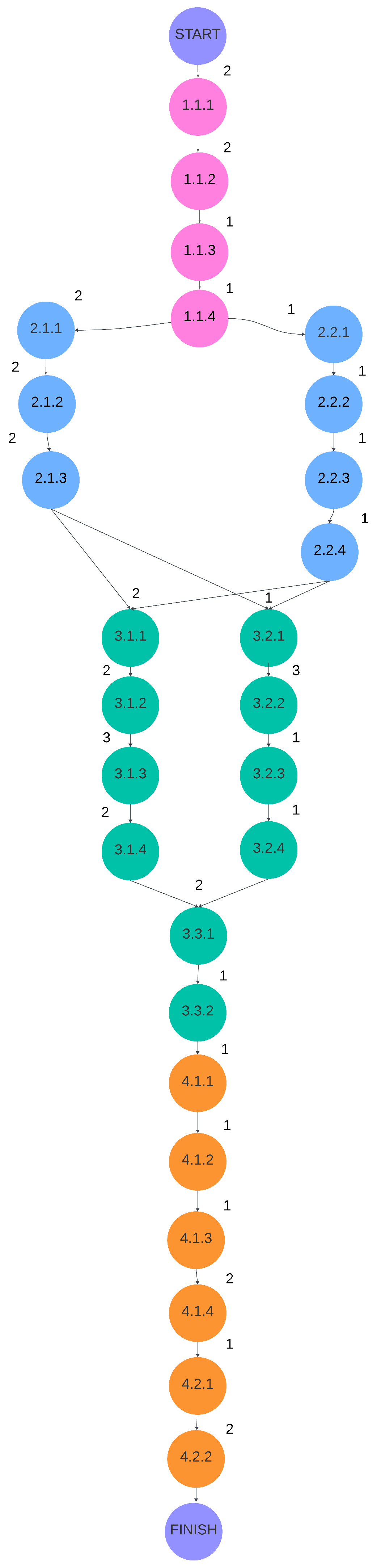
Step 4.1 Functional Testing

* Activity 4.1.1 Test task CRUD functionality
* Activity 4.1.2 Test calendar functionality
* Activity 4.1.3 Test notifications
* Activity 4.1.4 Make any adjustments needed based on testing

Step 4.2 User testing

* Activity 4.2.1 Simulate user testing session
* Activity 4.2.2 Make any adjustments based on testing

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Step** | **Activity** | **Time Estimate (days)** |
| **Phase 1: Analysis** | **Step 1.1: Gather Requirements** | Activity 1.1.1: Analyze existing task management applications | 2 |
|  |  | Activity 1.1.2: Make a list of potential stakeholders and their requirements | 2 |
|  |  | Activity 1.1.3: Document requirements | 1 |
|  |  | Activity 1.1.4: Finalize requirements | 1 |
| **Phase 2: Design** | **Step 2.1: Design UI** | Activity 2.1.1: Create wireframes | 2 |
|  |  | Activity 2.1.2: Create UI mockups | 2 |
|  |  | Activity 2.1.3: Finalize UI design | 2 |
|  | **Step 2.2: Design DB** | Activity 2.2.1: Define DB schema | 1 |
|  |  | Activity 2.2.2: Set up DB in XAMPP | 1 |
|  |  | Activity 2.2.3: Review and finalize DB design | 1 |
|  |  | Activity 2.2.4: Populate DB with test data | 1 |
| **Phase 3: Development** | **Step 3.1: Frontend Development** | Activity 3.1.1: Create HTML markup | 2 |
|  |  | Activity 3.1.2: Style with Bootstrap and CSS | 2 |
|  |  | Activity 3.1.3: Integrate jQuery as needed | 3 |
|  |  | Activity 3.1.4: Implement FullCalendar library | 2 |
|  | **Step 3.2: Backend Development** | Activity 3.2.1: Set up PHP | 1 |
|  |  | Activity 3.2.2: Write PHP for CRUD operations | 3 |
|  |  | Activity 3.2.3: Add user authentication | 1 |
|  |  | Activity 3.2.4: Integrate with DB | 1 |
|  | **Step 3.3: Integration** | Activity 3.3.1: Integrate frontend with backend | 2 |
|  |  | Activity 3.3.2: Test integration | 1 |
| **Phase 4: Testing** | **Step 4.1: Functional Testing** | Activity 4.1.1: Test task CRUD functionality | 1 |
|  |  | Activity 4.1.2: Test calendar functionality | 1 |
|  |  | Activity 4.1.3: Test notifications | 1 |
|  |  | Activity 4.1.4: Make any adjustments needed based on testing | 2 |
|  | **Step 4.2: User Testing** | Activity 4.2.1: Simulate user testing session | 1 |
|  |  | Activity 4.2.2: Make any adjustments based on testing | 2 |



Requirements

Functional

1. Users are able to register for an account using an email and password
2. Users can log in and out of their account
3. Users can reset their password as needed
4. Users can create a new task with a title, description, due date, status (completed/not-completed), notification timing, and priority.
5. Users can view, edit, and delete tasks.
6. Users can complete tasks
7. Users can categorize tasks by priority (high, medium, low)
8. Users can designate task type (category)
9. Users can organize tasks by category (ie work, school, project, family, health, etc)
10. Users can view tasks in a calendar view according to their due dates
11. Users can view calendar by day, week, and month
12. Users can receive notifications for upcoming tasks
13. User data is securely stored in the database
14. The application is responsive and works on various devices

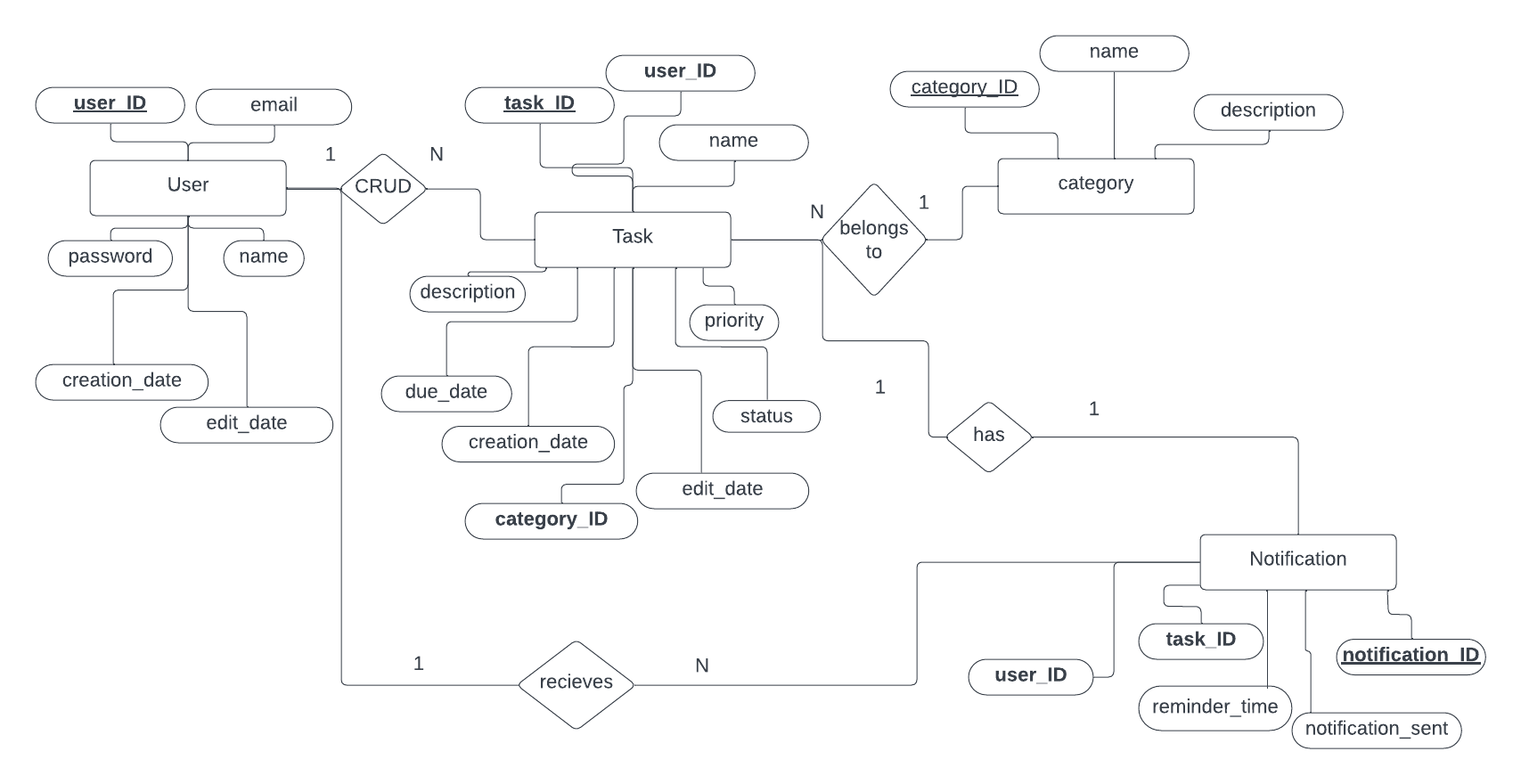
Non-functional

1. Application loads within 5 seconds
2. Task creation, deletion, and editing operations complete within 2 seconds
3. Calendar view updates real-time as tasks are created/edited/deleted
4. Strong password policies are implemented (minimum length and complexity)
5. UI is intuitive and easy to use
6. The app must be accessible to users with disabilities
7. The app will be compatible with major web browsers
8. The app works on Windows and macOS operating systems
9. Users can delete their account and data permanently

Entity Relationship diagram

* User CRUD (creates, reads, updates, deletes) task
* User can create many tasks
* Task can be CRUD (created, read, updated, deleted) by one user
* 1 user to many tasks
* Task belongs to category
* Category can have many tasks
* Many tasks to 1 category
* Task has a notification
* Notification is associated with one task
* 1 notification to.1 task
* User receives many notifications
* Notification is sent to one user
* 1 user to many notifications

Diagram on next page



**ASSIGNMENT 3**

Task Management App: Architecture Design

The user interacts with the front-end via the browser which triggers HTTP requests.

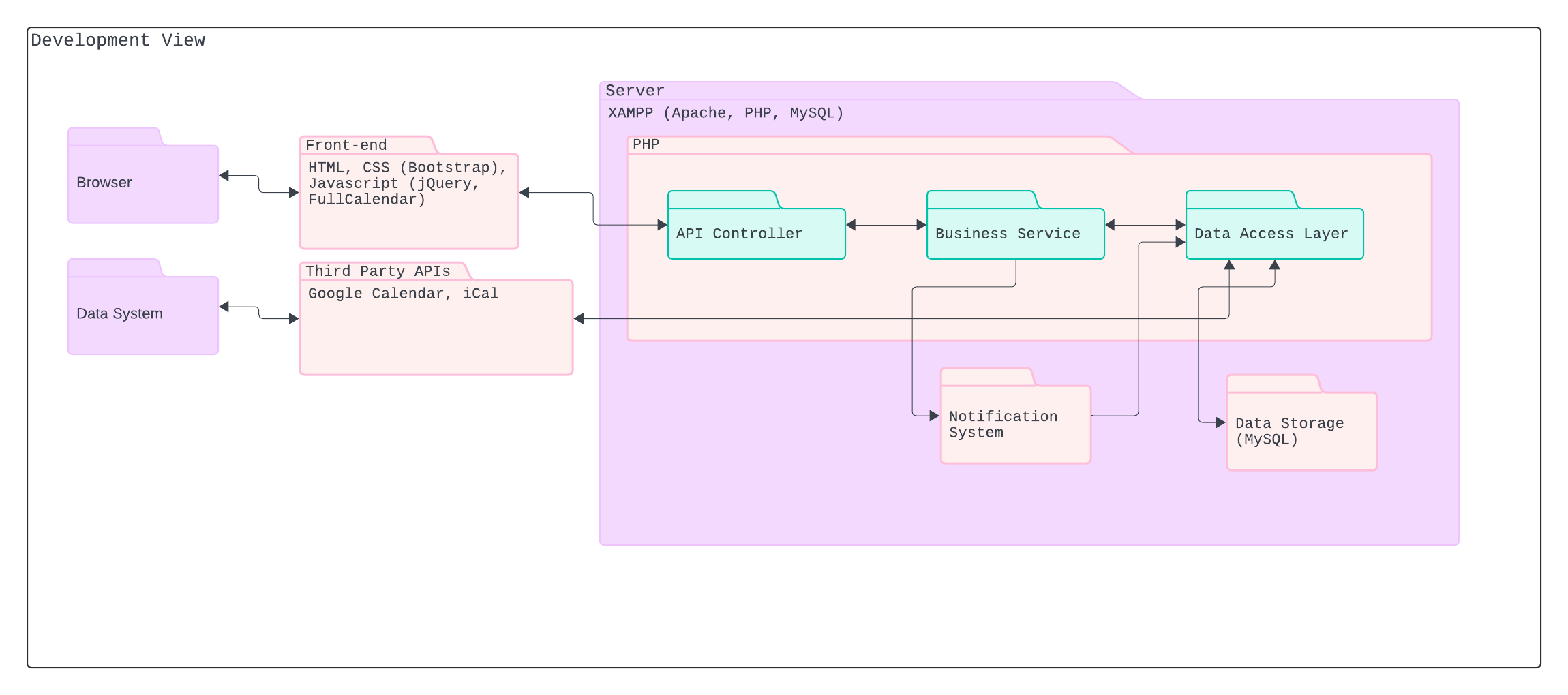
The API controller handles these HTTP requests and routes them to the business service.

The business service processes the requests and coordinates with the notification system or the data access layer depending on the requests.

The data access layer handles operations, such as performing SQL queries to retrieve or send data to MySQL and the data system (such as the third party APIs).

The notification system manages the notification services and sends information to the data access layer to store this data and send notifications.

The data system consists of third party APIs to synchronize tasks with Google Calendar or iCal.



The UML class diagram depicts several classes: User, Task, Category, Notification, Occurrence

1 User may have many tasks.

Many tasks may be related to 1 user.

1 task will have 1 notification.

1 notification belongs to 1 task.

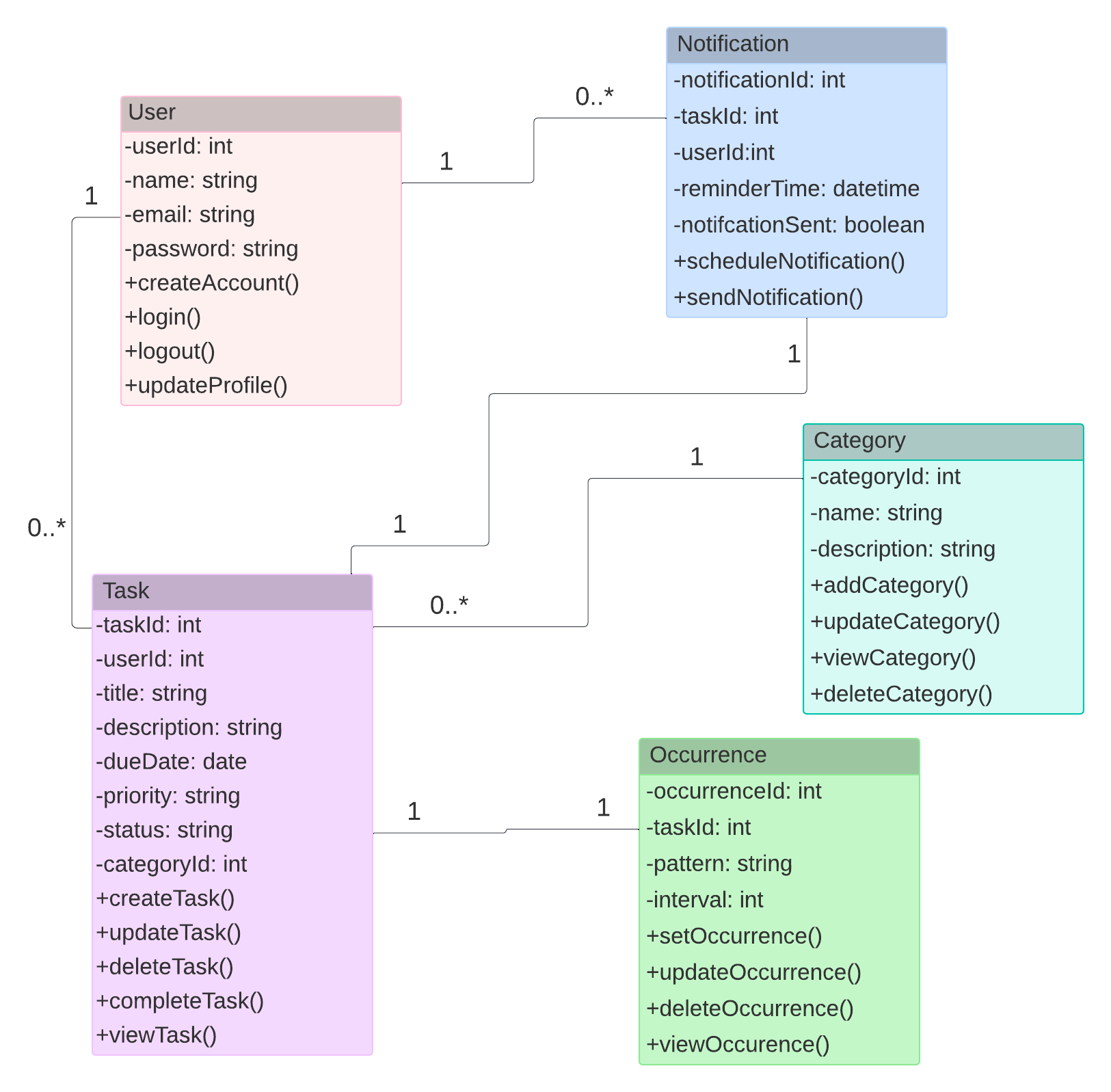
Many tasks belong to 1 category.

1 category may relate to many tasks.

1 task to 1 occurrence.

1 user may have many notifications

Many notifications may belong to 1 user



**ASSIGNMENT 4**

UserTest Unit testing

When I try to open in browser or run UserTests.php in the command line the first time it works, but if I try two times I see this error in the PHP logs:

PHP Fatal error: Uncaught PDOException: SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry 'testuser1@test.com' for key 'email' in /Applications/XAMPP/xamppfiles/htdocs/TaskMinder/php/User.php:40

Stack trace:

#0 /Applications/XAMPP/xamppfiles/htdocs/TaskMinder/php/User.php(40): PDOStatement->execute()

#1 /Applications/XAMPP/xamppfiles/htdocs/taskminder/tests/UserTests.php(12): User->createAccount('Test User1', 'testuser1@test....', 'password123')

#2 /Applications/XAMPP/xamppfiles/htdocs/taskminder/tests/UserTests.php(74): testCreateAccount()

#3 {main}

thrown in /Applications/XAMPP/xamppfiles/htdocs/TaskMinder/php/User.php on line 40

This error is ocuring because the email should be unique to each user, so when I try to create an account with a user name that already exists in the DB I get a constraint violation.

The fix for this is to add a validation check in my User->createAccount() method to check if the email exists before creating a new account.

Code needed to be uncommented after performing UserTests.php

Check if user already exists in the DB

$query = "SELECT COUNT(\*) as count from " . $this->table\_name . " WHERE email=:email";

$stmt = $this->connection->prepare($query);

$stmt->bindParam(':email', $email);

$stmt->execute();

$row = $stmt->fetch(PDO::FETCH\_ASSOC);

if ($row['count'] > 0) {

echo "Error: A user with this email already exists\n";

return false;

}

Next error is coming from the function testLoginCorrectCredentials() test. The user-> login() method is throwing this error

PHP Notice: session\_start(): Ignoring session\_start() because a session is already active in /Applications/XAMPP/xamppfiles/htdocs/TaskMinder/php/User.php on line 83

This is because the login() method starts a session without checking to see if a session already exists. The fix is to check if a session is started before starting a new session.

Code needed to be uncommented after performing UserTests.php

Check if session is already started before starting a session

if (session\_status() == PHP\_SESSION\_NONE) {

session\_start();

}

The next error is coming from the testLogOut() function. This is the error

PHP Notice: session\_start(): Ignoring session\_start() because a session is already active in /Applications/XAMPP/xamppfiles/htdocs/TaskMinder/php/User.php on line 102

Similar to the login error I shold check if the session is started before starting a session

Check if session is already started before starting a session

if (session\_status() == PHP\_SESSION\_NONE) {

session\_start();

}

**SUMMARY**

Unfortunately I was unable to get Docker to correctly serve the CSS files and PHP classes. I confirmed I have the correct set up for the Dockerfile and Apache config but I continued to get 404 errors. I eventually gave up and went back to XAMPP.

Additional development needed in the app with regards to notifications, category management, and enhancements to the dashboard for usability.