***SOFTWARE PROJECT FINAL REPORT***

Ben Siao (2848022), Alex Orban (2821580), Joe Norton (2837372), Allen Jiang (2846054)

4/20/2025

**Table of Contents**

1. Introduction: Page 2

2. Project Management Plan: Page 2

3. Requirement Specification: Page 5

4. Architecture: Page 6

5. Design: Page 7

6. Test Management: Page 13

7. Conclusions: Page 14

References: Page 15

**List of Figures**

Waterfall Model: Page 3

Use Case Model: Page 5

Architecture Model: Page 7

Static Model: Pages 9–12

Dynamic Model: Pages 9–12

**List of Tables**

Risk Analysis: Page 4

Traceability Requirements: Page 12–13

# 1. Introduction

1.1. Purpose and Scope

Our goal and objective were to create a chores app called ChoreHero which utilizes databases, HTML, and PHP to enforce user interfaces. SQL was used to create the database and to create tables (elements). PHP was used to enforce methods of searching and to establish the connection between the user interface and the database. The scope is that the user will start the software and then get presented to a sign in page where they can either log in or create a new user. The user interface presents an option to the user to add or complete a task. Once completing the task, the user face will show how many points you have after completing each chore.

1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

Our project products utilize the MAMP server for database management and the system uses MySQL queries to manage data retrievals and changes. The capabilities are briefly mentioned in the purpose and scope section. We were able to utilize PHP and HTML to create a user interface that allows users to interact with the database system. The user will be presented with the action to log in or to create a new account. Then the user will be able to add or complete tasks. This then increment's the score system for the specified user. This product could be used in a household where there is a reward system. For example, if you get 10 points in a week, you may be able to buy or redeem some reward worth that number of points. This might be useful for keeping a family organized and clean by giving family members an incentive to complete their household chores.

1.3. Structure of the Document

This document contains the introductions, project management plan, requirement specification, architecture, design, test management, and conclusion. By following this structure, readers and graders will get a sense of the structure of the project and hardware/software used in this project.

1.4. Terms, Acronyms, and Abbreviations

MAMP: Use to store the database

PHP: coding language to interact and connect to the database

HTML: Programming language to create a user interface.

# 2. Project Management Plan

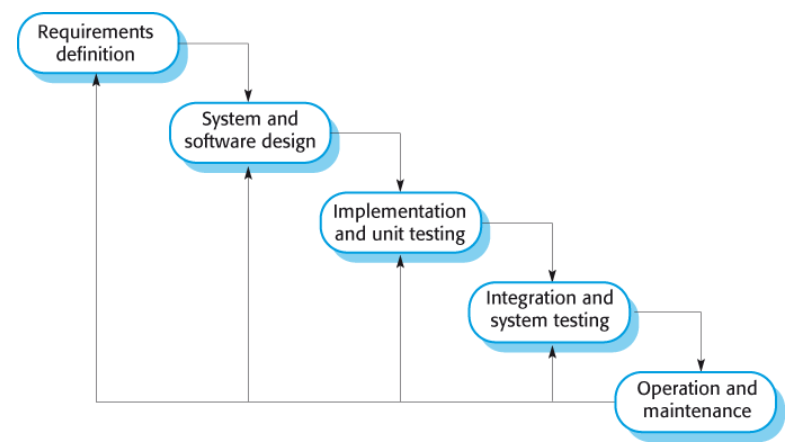
2.1. Project Organization

Our team followed role-based structure to ensure clear responsibilities and streamlined communication. Ben, Alex, Joe, and Allen worked together to complete the project, with each team member contributing to different areas such as backend development, frontend design, database integration, and system testing. Responsibilities were shared throughout the process, with weekly meetings held to track progress, resolve blockers, and plan tasks to ensure a functional final product.

* Ben – Backend Developer (PHP)
* Alex – Backend Developer, Tester (PHP)
* Joe – Frontend Developer (HTML/CSS/JS)
* Allen – Database and Backend Support (MySQL)

2.2. Lifecycle Model Used

We used the Waterfall Model. The project followed sequential phases including:



1. **Requirements Analysis and Definition**  
    We identified and documented the app’s core features which included chore assignment, user tracking, and rewards. These were based on team discussions and user needs.
2. **System and Software Design**  
    We designed a client-server architecture using MVC principles, outlining how the frontend, backend, and database would interact.
3. **Implementation and Unit Testing**  
    Each module was developed and tested individually, including chore creation, user login, and point tracking functionalities.
4. **Integration and System Testing**  
    All components were combined and tested as a complete system to ensure they met the project requirements and worked seamlessly.
5. **Operation and Maintenance**  
    We prepared the app for future use and updates by documenting setup steps and planning for usability and feature enhancements.

2.3. Risk Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk Name** | **Probability** | **Impact** | **Migration Strategy** |
| Technical Difficulties | Medium | High | Component testing, debugging tools, weekly code reviews |
| Security Concerns | Medium | High | Secure auth, input validation |
| Performance Issues | Low | Medium | |  | | --- | | Optimize queries, performance monitoring | |
| Data Loss | Low | High | Backups, rollback mechanisms, restricted data access |
| User Engagement | Medium | Medium | feedback loops, rewards/incentives |
| Scope Creep | Medium | Medium | Define scope early, use change control, prioritize essential features |

2.4. Hardware and Software Resource Requirements

* **Hardware:**
  + Laptops/desktops for each team member
  + Server for MAMP deployment (local testing)
* **Software:**
  + MAMP (MySQL, Apache, PHP)
  + HTML/CSS/JavaScript
  + PHP

2.5. Deliverables and schedule

|  |  |
| --- | --- |
| **Deliverable** | **Due Date** |
| Project Plan  Software Requirements Specification  Software Design Specification | 2/22  3/23  3/30 |
| System Design Specification | 3/30 |
| Frontend Prototype | 4/6 |
| Backend API & Database Integration | 4/6 |
| Testing Report | 4/28 |
| Final Project (working app + documentation) | 4/28 |
| Project Presentation | 4/28 |

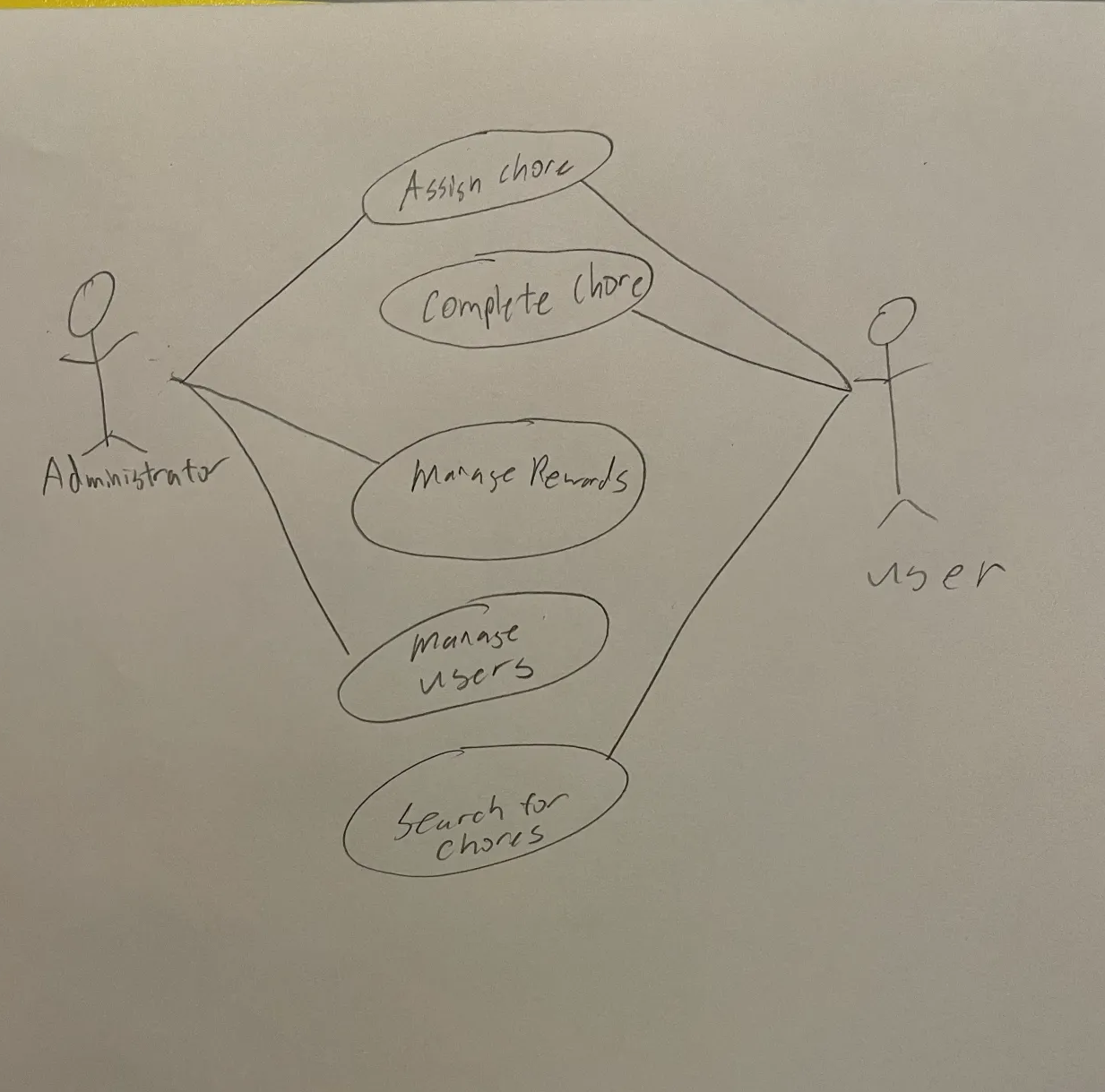
# 3. Requirement Specifications

3.1. Stakeholders for the system

The stakeholders for the system are the developers and the end users. These end users could be family members or other household users. Investors could also be a stakeholder for the system if we present this as an actual application in the real world.

3.2. Use cases

3.2.1. Graphic use case model



3.2.2. Textual Description for each use case

- Assigning a chore: all users can do this. However, we could change it to admin roles only in the future for security purposes. This use case involves assigning a user a chore. The details of the chore will be assigned including the name and a deadline.

- Complete chore is where a chore is marked completed. This will be done by the user when they mark each chore as complete. This changes the status of the database to “complete” when the chore is completed.

- Mangement rewards will be done by the administrator to compute the points for each specific chore and tracks each user to how many rewards points they have based on the number of chores completed.

- Managing users is done by the admin (the user with database access) who can remove new users, remove user accounts. They can also modify each user’s accounts.

- Searching for chores is done by the user to see what chores need to be completed. Chores show up after assigning them.

3.3. Rationale for your use case model

The use case model we have designed above shows the interactions between users and the system to manage chores and rewarding based on amount for chores. The rationale behind this use case model is based on the scope and the purpose of our project. It is structured to show the organization of assigning and tracking tasks. It also correctly shows the management of the users and how rewards are distributed.

3.4. Non-functional requirements

Some of the nonfunctional requirements are security, scalability, and time restraints. The project should make sure that the system hashes the password for each user to prevent unauthorized access to user accounts and their data. We also have to make sure that the system will allow multiple concurrent users at the interface. The database must allow a signification number of registered users. The timing restraints must be satisfied so that the user would be able to search for chores that's been assigned.

# 4. Architecture

4.1. Architectural style(s) used

The application will follow a Client-Server architecture with Model-View-Controller (MVC) principles. The client (browser) interacts with the server using HTTP requests, while server-side logic handles data operations and view rendering.

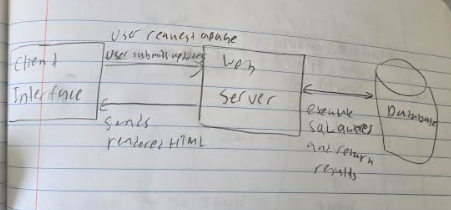
4.2. Architectural model (includes components and their interactions)

**Components:**

* Client Interface (HTML/CSS/JavaScript) – Displays chore list, search, and status
* Web Server (PHP) – Handles requests and routes them to the correct logic
* Database (MySQL) – Stores users, chores, assignment logs, and reward points

**Interactions:**

1. User requests a page (e.g., chore list)
2. Server retrieves data from the MySQL database
3. Server sends rendered HTML back to the client
4. User submits updates (e.g., marks chore as complete), which are sent back to the server for processing



4.3. Technology, software, and hardware used

* Frontend: HTML, CSS, JavaScript
* Backend: PHP
* Database: MySQL
* Environment: MAMP Server
* Version Control: GitHub
* Hardware: Personal computers for development, localhost for testing

4.4. Rationale for your architectural style and model

The Client-Server and MVC architecture offer separation of concerns and scalability, which is ideal for a chore management web app. The client-server model ensures the user interface stays clean and responsive, while backend logic and database operations remain centralized and secure. MVC helps with modular development and easier debugging.

# 5. Design

5.1. User Interface design

The ChoreHero interface is a web-based application using HTML, CSS, and PHP to manage user interaction and backend logic. The UI is intentionally simple and intuitive, catering to all ages in a household environment.

Main User-Facing Pages:

* login.php: Allows users to securely log in. Displays validation errors when credentials are incorrect.
* index.php: The homepage/dashboard displaying user-specific chore data and options to navigate or log out.
* add-task.php: Provides a form to add a new chore with task name and optional due date.
* display-chores.php: Displays a list of chores assigned to the user with an option to mark as complete.

Backend/Helper Scripts (not visible in UI):

* submit-chores.php: Updates the status of chores and increments user points.
* db\_connect.php: Handles MySQL database connection logic.
* logout.php: Manages session destruction and redirects users.

The UI follows standard design principles:

* Consistency: Uniform color palette and layout across pages.
* Clarity: Clear labels like “Add Chore” or “Mark as Complete”.
* Accessibility: Keyboard navigation and strong contrast.

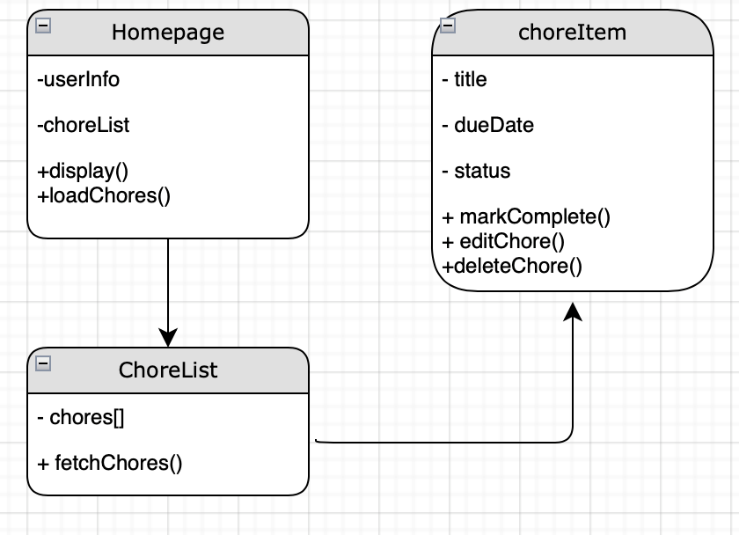
5.2. Components design (static and dynamic models of each component)

The system is divided into three key components:

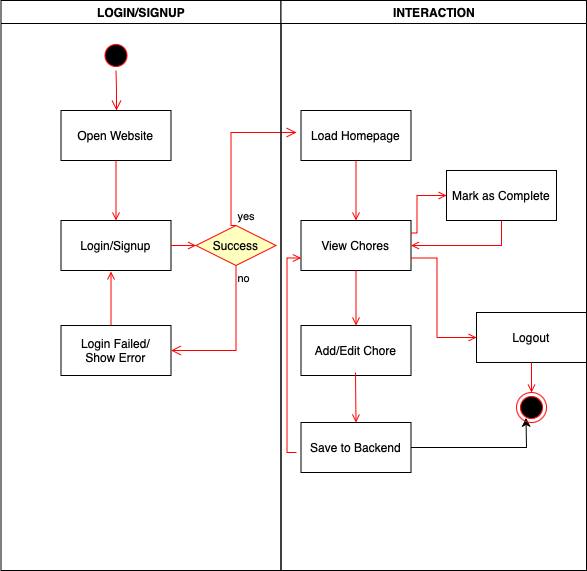
* Frontend (HTML/CSS): Manages the UI/UX.
* Backend (PHP): This handles user requests and business logic.
* Database (MySQL): Stores user and chore-related data

**1. Frontend Component description**

* Inputs:
* Name, Email, Password, Confirm Password
* Task Name, Description, Due Date
* Mark task as complete
* Outputs:
* Message: Made account successfully
* Display tasks on homepage
* Confirmation messages
* Error messages
* Exceptions:
* Server errors
* Page not found
* Unsuccessful authentication
* Static model:

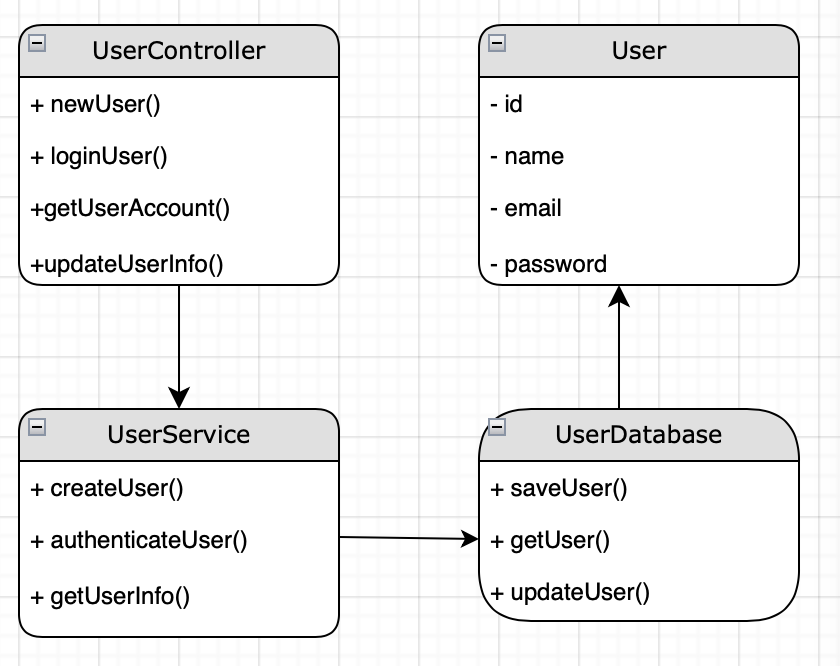


* Dynamic model:

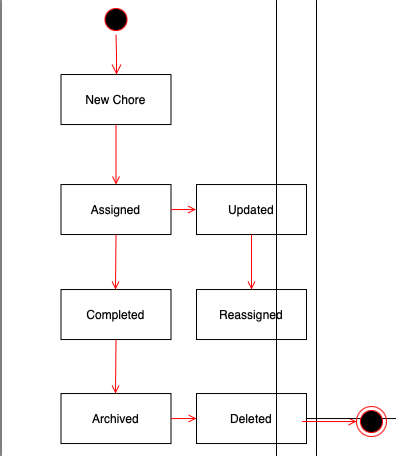


**2. Backend Component description**

* Inputs:
* Signup: {name, email, fname, lname, password}
* Create Chore: {id, choreName, Description, dueDate}
* Use chore ID as identifier to change and delete chore or mark as complete
* Outputs:
* {id, title, status(notComplete, Complete), dueDate}
* Exceptions:
* Invalid Input, Unauthorized Access, Task Not Found
* Static model:

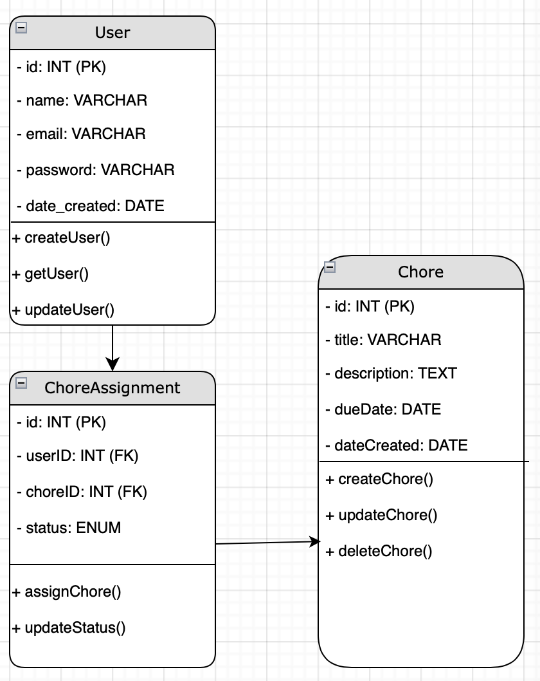


* Dynamic model:

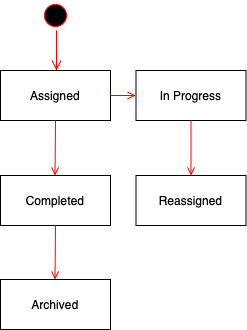


**3. Database Component Description**

* Inputs:
* INSERT INTO users (name, email, password, created\_at) VALUES ('John Doe', '[user@example.com](mailto:user@example.com)', 'password', NOW());
* INSERT INTO chores (title, description, assignee\_id, due\_date, status, created\_at) VALUES ('[choreName]',’[description]', [id#], '[dueDate]', '[status]', NOW());
* Outputs:
* { "id": 090, "name": "John Smith", "password: "password" }
* {"id": 1, "title": "Vacuum Living Room", "description": "Vacuum the carpet", "due\_date": "2025-04-01", "status": "pending"}
* Exceptions:
* Duplicate Task
* Task Not Found
* Static model:



* Dynamic model (different statuses of a chore in the database):



5.3. Database design

* User Table: id, name, email, password, score, created\_at
* Chore Table: id, title, description, user\_id, status, due\_date, created\_at
* Reward Table: rewardID, rewardName, choreID, userID, rewardPoints .

These structures support primary and foreign key relationships. This allows for accurate chore-user mapping and reward tracking.

5.4. Rationale for your detailed design models

The system was designed to be clear and easy to maintain. Each part of the project represents something from real life, like users, chores, and rewards. The database was structured with straightforward relationships so that tasks are correctly linked to users and point tracking is accurate. Keeping the frontend and backend separate makes it easier to update one side without affecting the other. For example, if we need to change how chores are shown to the user, we can do that without touching the database logic.

The diagrams we created helped us picture how someone would use the app, starting from logging in and ending with completing a chore. This made it easier to plan where the logic should go and how everything would connect. These models helped keep our work organized and made the development process more manageable.

5.5. Traceability from requirements to detailed design models

|  |  |
| --- | --- |
| **Requirement** | **Design Element** |
| User Registration | User table, login.php, UserController |
| Chore Management | Chore table, add-task.php, display-chores.php |
| Task Completion & Points | submit-chores.php, score in User table |
| Secure Authentication | password\_hash() in login.php, session management |
| UI Navigation | Navigation bar across all HTML views |
| Data Storage | MySQL via db\_connect.php |

# 6. Test Management

6.1. A complete list of system test cases

1. Login/Signup with valid inputs
2. Login/Signup with invalid inputs
3. Add new task
4. Delete task
5. Edit task description
6. View all tasks
7. Task sorting by date
8. Input empty task

6.2. Traceability of test cases to use cases (Test Case Affected)

1. Login/Signup (1, 2)
2. Add task (3, 8)
3. Edit task (4)
4. Delete task (5)
5. View task (6)
6. Sort tasks (7)

6.3. Techniques used for test case generation

1. Test one input at a time
2. Test if valid inputs are taken successfully and invalid input taken unsuccessfully
3. Smooth transition between login window, task dashboard, and add task window
4. Test each use case

6.4. Test results and assessments (how good are your test cases? How good is your software?)

All test cases were passed successfully on several search engines and devices. All user tasks were tested and left no errors or unsuccessful functionality. Error messages were displayed in the case of invalid inputs. Tests that isolated each function make sure all backend functions worked properly.

6.5. Defects reports

Past dates input successfully when it should be invalid. Invalid emails were input successfully. Sorting was wrong when a task was added without a date.

# 7. Conclusions

7.1. Outcomes of the project (are all goals achieved?)

The main goals of ChoreHero were achieved:

* Users can register, log in, and manage chores.
* Tasks can be added and marked as complete.
* User scores are updated and stored persistently.
* The interface functions across devices and offers real-time feedback.
* The system successfully integrates HTML, PHP, and MySQL for a seamless experience.

Some features that were in consideration, like a full rewards store or a shared household leaderboard, were set aside for future development.

7.2. Lessons learned

This project provided valuable experience with the full software development process, from planning and design through implementation and testing. We saw how important it is to define team roles clearly and maintain regular communication to keep development on track. Following a structured development model helped us stay organized and made it easier to meet our deadlines.

We also gained practical skills in building a full-stack application using PHP and MySQL. Connecting the frontend with the backend required careful attention to user input handling, session management, and database integration. We also learned how to troubleshoot unexpected behavior and use debugging techniques to identify and resolve issues efficiently.

User experience turned out to be just as important as technical functionality. We noticed that even small improvements in layout or feedback messages made the app much easier to use. Ultimately, this project taught us how important it is to plan around real user behavior, prioritize clarity in both design and functionality, and test thoroughly to make sure each part of the system works as expected.

7.3. Future development

* Add a rewards page to let users redeem points
* Implement household grouping and admin roles
* Develop chore editing and deletion functionality with validation
* Add email reminders for upcoming or overdue chores
* Transition to a cloud-based server for remote access and scalability
* Improve security by enforcing stricter validation and user permissions

# References

*Database SQL Reference*. (n.d.). Docs.oracle.com. <https://docs.oracle.com/cd/B19306_01/server.102/b14200/toc.htm>

GeeksforGeeks. (2024). *GeeksforGeeks | A computer science portal for geeks*. GeeksforGeeks. <https://www.geeksforgeeks.org/>

LucidChart. (2017). *Online Diagram Software & Visual Solution | Lucidchart*. Lucidchart.com. <https://www.lucidchart.com/>

*PHP: Language Reference - Manual*. (2019). Php.net. <https://www.php.net/manual/en/langref.php>

phpMyAdmin. (2019). *Bringing MySQL to the web*. PhpMyAdmin. <https://www.phpmyadmin.net/>

W3Schools. (1999). *W3Schools online web tutorials*. W3schools.com; W3Schools. <https://www.w3schools.com/>