Eric LaBerge

Fo3126

CSC 4996

To Do List Web Application

**Functional Requirements:**

|  |
| --- |
| **FR01: Add and Remove Items** |
| The user should be able to remove and add tasks to the To Do list |

|  |
| --- |
| **FR02: View current tasks** |
| The user should be able to view the current list of tasks stored on the database |

|  |
| --- |
| **FR03: Web Interface** |
| All user-application interaction should be able to be achieved through a web application interface |

**Desired Requirements:**

|  |
| --- |
| **DR01: Login** |
| The application should require credentials to access the To Do list |

|  |
| --- |
| **DR02: Timestamp** |
| The tasks should be timestamped and the timestamps should be displayed in the same table |

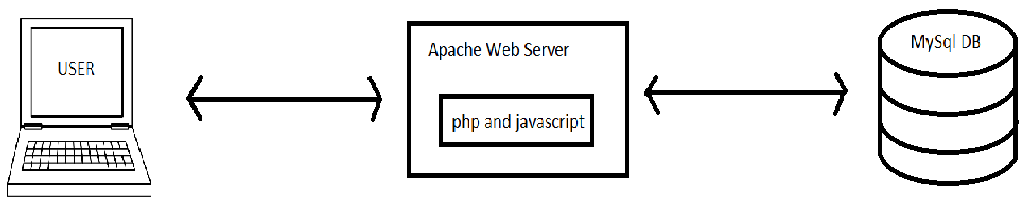
**Non-Functional Requirements**

|  |
| --- |
| **NR01: XAMPP Stack** |
| The web app should be built and deployed using the XAMPP software stack |

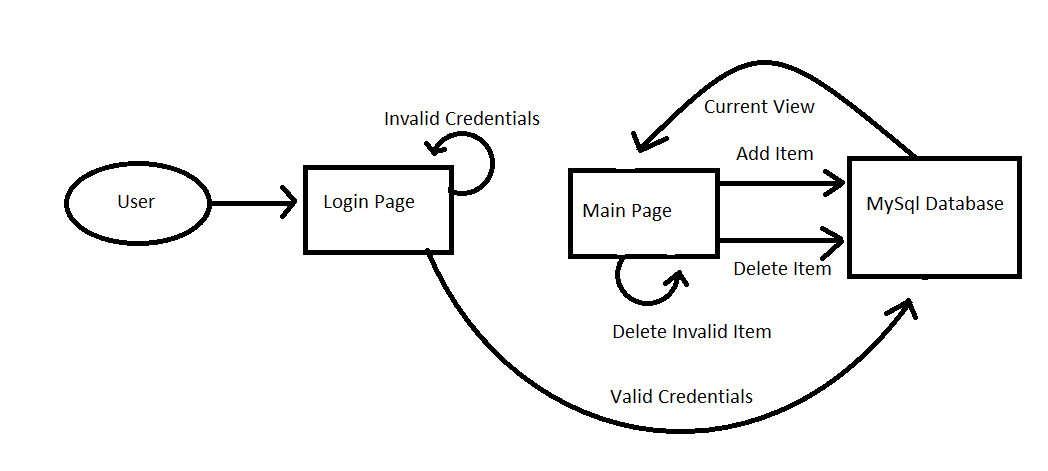
|  |
| --- |
| **NR02: Usability** |
| The software should be relatively easy to learn/use |

|  |
| --- |
| **NR03: Stability** |
| User action should not cause a crash/lapse in service of the application |

**System Architecture Diagram:**

****

**Data Flow Diagram:**

****

**Use Case:**

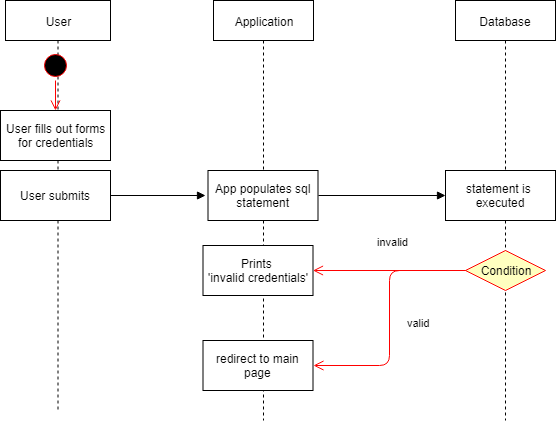
|  |  |
| --- | --- |
| **ID** | **Login** |
| 1. **Description** | This use case describes how a user can use the web application to present credentials in order to access the database |
| 1. **Actors** | Primary: User  Secondary: Web application, Database |
| 1. **Precondition** | 1. The user can access the web application 2. The web application is connected to the database |
| 1. **Basic flows** | 1. The user inputs credentials into the designated fields 2. The user submits these credentials 3. These credentials are leveraged against the user table contained in the database 4. The credentials are valid, and the user is redirected to the main page that contains the To Do list view |
| 1. **Alternative Flows** | 1. The user inputs credentials into the designated fields 2. The user submits these credentials 3. These credentials are leveraged against the user table contained in the database 4. The credentials are invalid 5. The web page informs the user the credentials entered are invalid 6. The user may try again |
| 1. **Post Conditions** | **Successful Completion:**  The user has access to the page to add or remove entries in the To Do list  **Failure Condition:**  The user does not proceed and can attempt to enter valid credentials again |
| 1. **Special requirements** | 1. The user must be notified their entered credentials were not valid if that is the case 2. The Password characters are hidden for security reasons |

|  |  |
| --- | --- |
| **ID** | **Remove Task** |
| 1. **Description** | This use case describes how a customer removes a task from the To Do list |
| 1. **Actors** | Primary: User  Secondary: Web application, Database |
| 1. **Precondition** | 1. The user can access the web application 2. The web application is connected to the database 3. The user has successfully logged in |
| 1. **Basic flows** | 1. The user enters the id of the desired task to be removed into the form 2. They click the submit button labeled ‘delete’ 3. The contents of the form are saved 4. An SQL statement is executed using the data from the form 5. The database is updated 6. The page is reloaded with the updated view |
| 1. **Alternative Flows** | 1. The user enters an id that doesn’t exist 2. The SQL statement is executed but doesn’t change the table |
| 1. **Post Conditions** | **Successful Completion:**  The requested task has been removed from the database and current view  **Failure Conditions:**  The requested task id doesn’t exist, and therefore cannot be removed |
| 1. **Special requirements** | N/A |

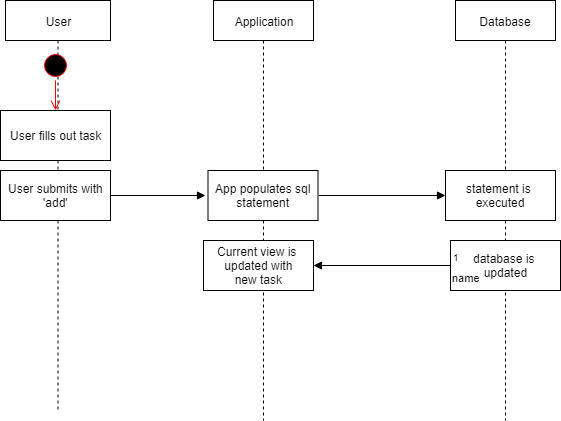
|  |  |
| --- | --- |
| **ID** | **Add Task** |
| 1. **Description** | This use case describes how the user uses the web application to add a task to the To Do list |
| 1. **Actors** | Primary: User  Secondary: Web application, Database |
| 1. **Precondition** | 1. The user can access the web application 2. The web application is connected to the database 3. The user has successfully logged in |
| 1. **Basic flows** | 1. The user enters their desired task into the form 2. They click the submit button labeled ‘add’ 3. The contents of the form are saved 4. An SQL statement is executed using the data from the form 5. The database is updated 6. The page is reloaded with the updated view |
| 1. **Alternative Flows** | There can be blank tasks, so there are no alternative flows |
| 1. **Post Conditions** | **Successful Completion**  The specified task is added to the database and the current view  **Failure Conditions:**  There are none, blank tasks are treated as valid |
| 1. **Special requirements** | N/A |

**Sequence Diagram:**

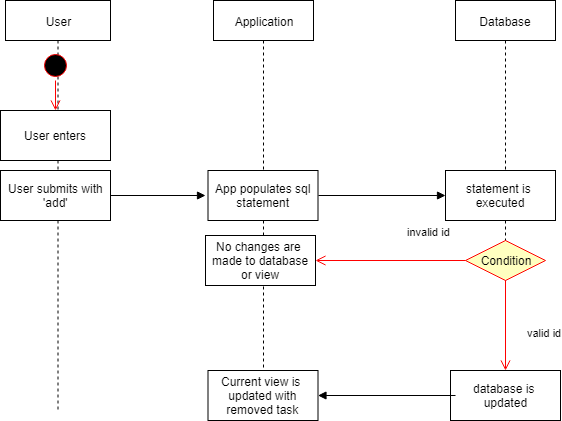
**Login:**



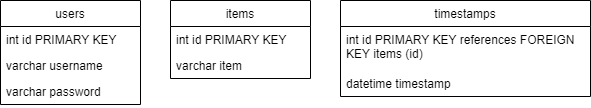
**Add:**



**Remove:**



**Database Diagram:**



**Test Cases:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case** | **Condition** | **Variables** | **Expected** | **Actual** | **Pass/Fail** |
| **1** | Invalid credentials | User: Eric  Pass: LaBerge | Failed login | Failed login | Pass |
| **2** | Bypass login | Navigate straight to localhost/project/post\_login.php | Redirect to index.html | Redirect to index.html | Pass |
| **3** | Delete nonexistent item | Remove item id 6 | No change | No change | Pass |