Omer Khan

1/14/18

CSC 4996

To-Do List App Documentation

**Functional Requirements**

1. The user can add a new task to the list, by entering information such as Task Name, a pre-defined option for task status (completed, pending, late, started), and a due date for the task. These changes are automatically updated in the database.

2. When a user clicks one of the filtering options, only tasks of those type are shown. For example, if a user chooses “Pending”, only pending tasks are shown on the list, and the rest are hidden. The filter options also display how many of each type of task are currently available.

3. A user can easily delete a task by selecting the ‘x’ option at the end of the row.

4. Changes made on the app (addition/deletion of tasks) are automatically reflected in the “todo” database.

5. A user cannot add any tasks with missing information (Task Name / Due Date / Status). When they attempt to do so, they receive an error message.

6. User can view how many tasks are available in the system, and they can see how many of each corresponding status as well.

**Non-Functional Requirements**

1. Usability: Due to the simple nature of the application, it is very intuitive to user. It is easy for a new user to see how to add a task because of headings and comments, and barrier of entry to use is very low.

2. Reliability: The application passes all required tests of required functionality. There is definitely room to test aspect such as performance more thoroughly however.

3. Performance: Performance so far is decent, there is a second or so lag between adding or removing a task and seeing the update on the page. There is room for improvement as far as making actions instantaneous.

4. Documentation: This guide acts as documentation for the project, allowing future users ease when attempting to pick up.

5. Cost: This project is free to use for any user. All they need to do is to install the devserver from EasyPHP, which is free to download.

6. Testability: The app can easily be tested using the test cases provided later on in this document to ensure functionality.

**System Architecture Diagram**

Makes Request (Add/Delete)

Response

User (UI)

Query

SQL

Database

Authentication

Web Server

**Data Flow Diagram**

Process Task

Add task to table

SQL Table

Add task

Get Task Information

Add task to table

Delete Task

Get Task to be deleted.

**Use Case**

1.

Name: Add Task

Brief Description: A user may add a task to the do list.

Actors: The user, database, and web application.

Preconditions: User has included all required information, and is able access the web page.

Basic Flow: User navigates to the web page and enter the required task information. Then, they select the ‘+’ symbol. This information is sent as a query by the web application to the SQL database, which stores the task with a unique ID. The web page is refreshed and the table view now includes newly added task.

Exception Flows: User does not include all required information and therefore request is not successfully sent.

Post Condition: New task is now stored in the SQL database, and table view displays newly added task.

2.

Name: Delete Task

Brief Description: A user may delete a task from the list.

Actors: The user, database, and web application.

Preconditions: User is able access the web page.

Basic Flow: User navigates to the web page and views the current tasks in the table. They select ‘x’ option next to a row for a task. The deletion request by the web app to the database, which runs a delete statement on the table. Web page is refreshed, and task is not displayed on table view anymore.

Exception Flows: User accidentally deletes the wrong task.

Post Condition: New task is now deleted from SQL database, and table view no longer shows task.

3.

Name: View Tasks

Brief Description: A user may view all inputted tasks.

Actors: The user, database, and web application.

Preconditions: User is able to access web page.

Basic Flow: User navigates to the web page. On load, the web page queries database and displays all current tasks in table. User can view task, due date, and the current status.

Exception Flows: There are no tasks currently in the database.

Post Condition: User can view and interact with all current tasks.

4.

Name: Pending Task Filter

Brief Description: A user may use the “pending task” filter to view all pending tasks.

Actors: The user, database, and web application.

Preconditions: User is able to access web page.

Basic Flow: User navigates to the web page. On load, the web page queries database and displays all current tasks in table. User selects “Pending Tasks”. Using client side jQuery, the web page hides all non-Pending tasks. User now only sees currently pending tasks.

Exception Flows: There are no pending tasks in the system.

Post Condition: User can filter and interact with all pending tasks.

5.

Name: Late Task Filter

Brief Description: A user may use the “late task” filter to view all late tasks.

Actors: The user, database, and web application.

Preconditions: User is able to access web page.

Basic Flow: User navigates to the web page. On load, the web page queries database and displays all current tasks in table. User selects “Late Tasks”. Using client side jQuery, the web page hides all non-Late tasks. User now only sees currently late tasks.

Exception Flows: There are no late tasks in the system.

Post Condition: User can filter and interact with all late tasks.

6.

Name: Completed Task Filter

Brief Description: A user may use the “completed task” filter to view all completed tasks.

Actors: The user, database, and web application.

Preconditions: User is able to access web page.

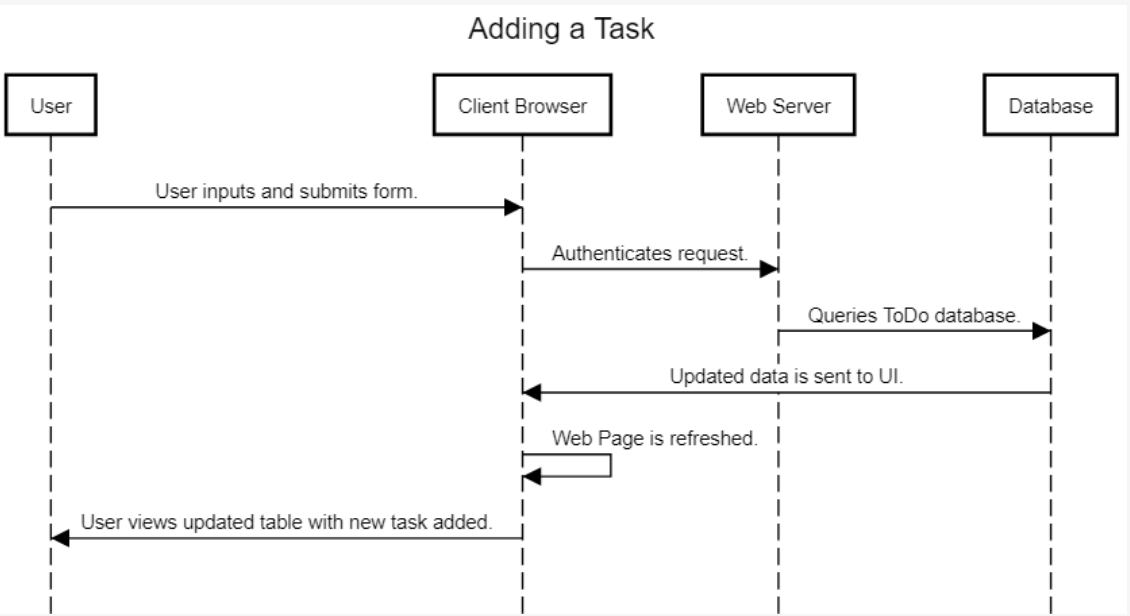
Basic Flow: User navigates to the web page. On load, the web page queries database and displays all completed tasks in table. User selects “Completed Tasks”. Using client side jQuery, the web page hides all non-Completed tasks. User now only sees currently completed tasks.

Exception Flows: There are no completed tasks in the system.

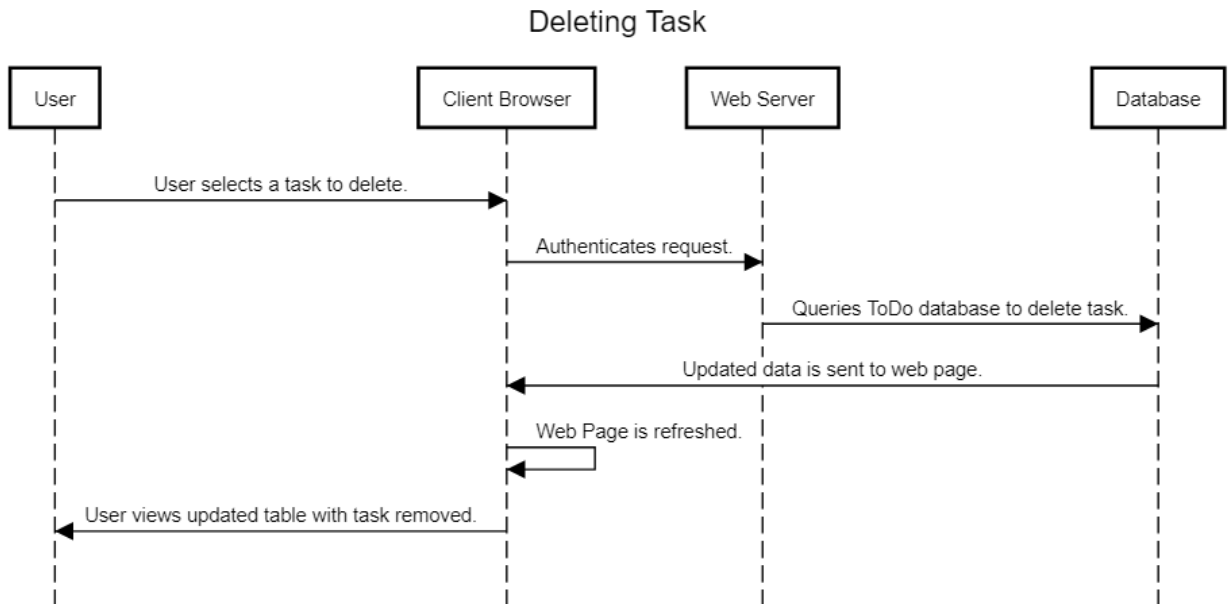
Post Condition: User can filter and interact with all completed tasks.

**Sequence Digram**

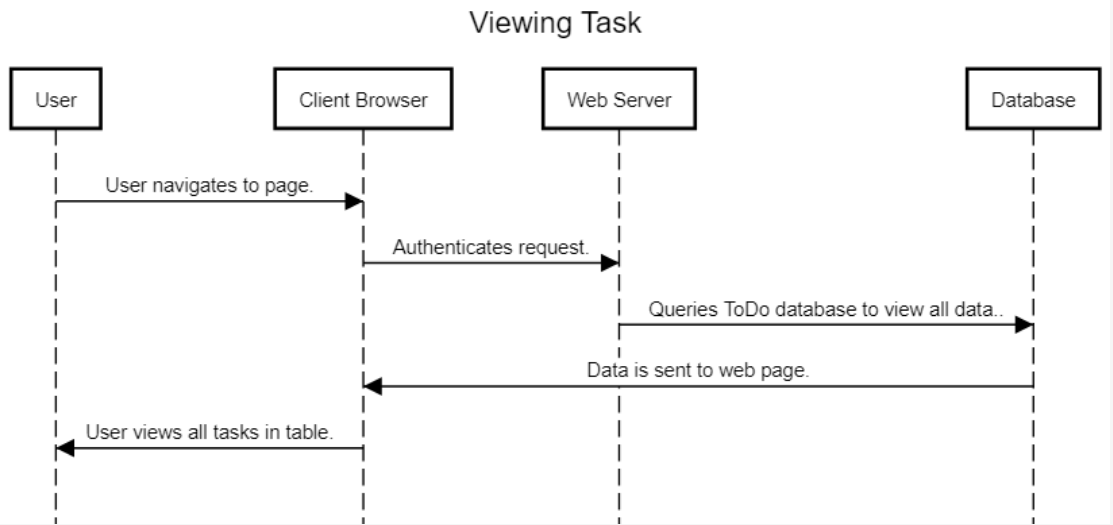
1. Adding a Task



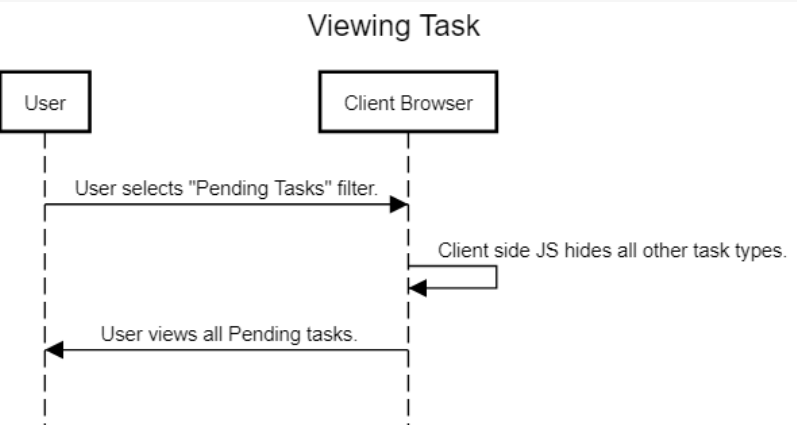
2. Deleting a Task



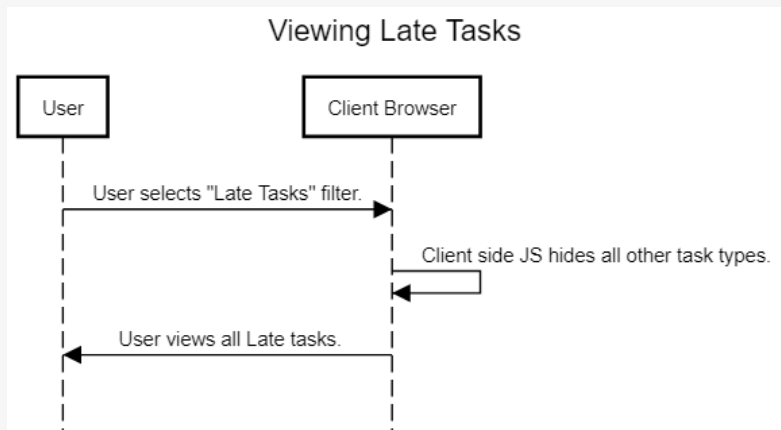
3. Viewing Tasks



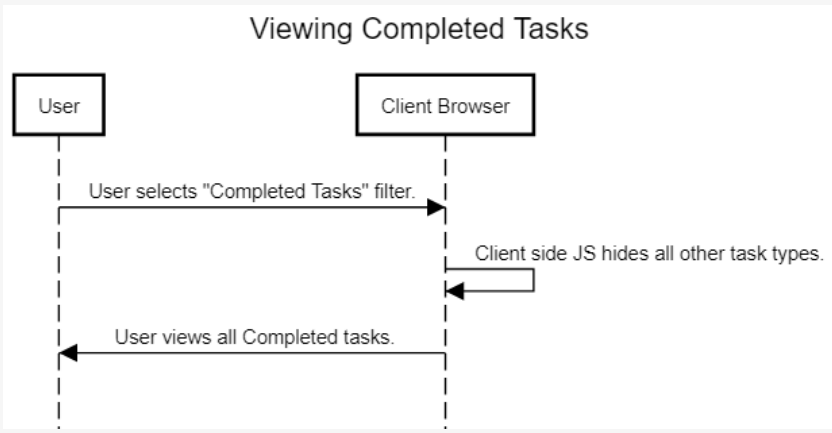
4. Filter Pending Tasks



5. Filter Late Tasks



6. Filter Completed Tasks



**Database Design**

**Tasks Table**

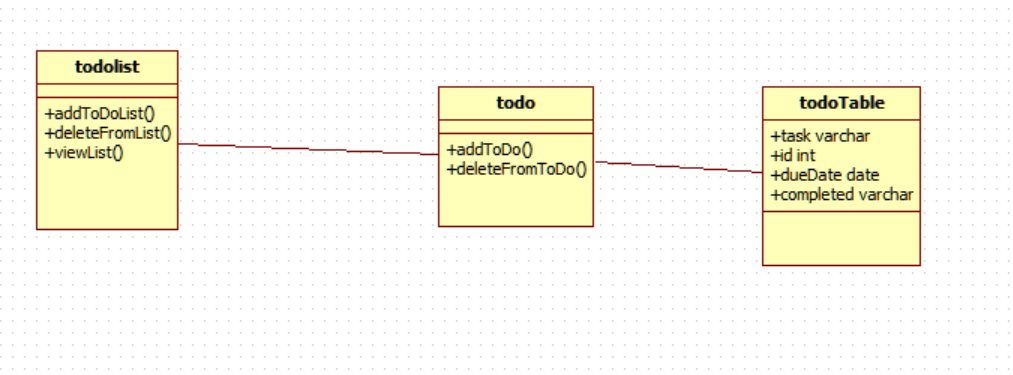
task varchar(200)

completed varchar(50)

dueDate date

id int auto\_increment primary key

**Class Diagram**



**Test Case**

|  |  |  |
| --- | --- | --- |
| Test Case Task | Steps | Expected Results |
| Add a Task | 1. Navigate to web page.  2. Fill out all required fields.  3. Press ‘+’ icon to add task.  4. Wait for page to refresh and verify task appears on list. | The new task should be listed on the table, and also in the SQL database. |
| Delete a Task | 1. Navigate to web page.  2. On task list, click ‘x’ icon next to a given task.  3. Wait for page to refresh. | The task should not appear on the table, and it should also have been removed from SQL database. |
| View all Tasks | 1. Navigate to web page.  2. View Table of Tasks. | A table showing all of the tasks should be visible. |
| Filter Task | 1. Navigate to web page.  2. View list of tasks available.  3. Click on a filter of tasks. | Only tasks of the selected type should be visible. |
| Number of Tasks | 1. Navigate to web page.  2. View number of tasks for each status type. | Ensure that numbers equal the tasks displayed in the table. |
| Cannot add task with missing fields | 1. Navigate to web page.  2. Attempt to add a task without filling out form.  3. Note result. | Should prompt an error message when trying to submit. |

**Time Breakdown**

Name of the task: Coding of Application

Estimated No of hours: 6 hours

Actual time spent: 8 hours

Notes: This included the developing of the front-end (HTML/CSS/JS), back-end (PHP/SQL), and deploying the application.

Issues: Development didn’t take too long, it took me a little while to deploy the server and the application since I hadn’t ran a server on my machine in a while.

Name of the task: Documentation

Estimated No of hours: 2 hours

Actual time spent: 2 hours

Notes: Writing up documentation guide and deployment guide.

Issues: Documentation took about as long as I expected. I had to research some of the diagrams I wasn’t familiar with.