B9IS109 Web Development for Information Systems

Module Lecturer: Obinna Izima Carzone

Student Name: Mahendra Vajja

Student No : 10639742

Contents

Table of Figures

1.Introduction

The Todo app is a simple and powerful tool designed to help people organize and manage their work efficiently. Given the hectic nature of modern life, it's easy to get overwhelmed with countless responsibilities, tasks, and projects. The Todo app serves as a digital to-do list, allowing users to capture, prioritize, and track their tasks in one place.

The main purpose of the Todo app is to increase productivity and time management by providing a clear and structured way to track tasks and make sure nothing slips through the cracks. Users can create tasks, set due dates, add descriptions, and even mark tasks as complete when completed. Whether it's a personal task, a work-related task, or a reminder for an important event, the Todo app serves as a trusted companion to help users stay organized and focused. central. Many Todo apps come with extra features like task categorization, notifications, collaboration, and cross-device synchronization, making it a must-have for people of all ages. social class.

In this fast-paced, always-connected world, having the Todo app at your fingertips can make a huge difference in staying productive, reducing stress, and achieving your personal and business goals. profession effectively."

2.Research and Planning

1. Define Purpose and Scope: Define the main purpose of your Todo application. Is it for personal use, team collaboration, or both? Decide on the essential features and functionality you want to include.

2. User search: Conduct user research to understand the needs and pain points of the target audience. Collect feedback through surveys, interviews, or user testing sessions. Identify the features they need and the problems they face with existing Todo applications.

3. Competitor analysis: Analysis of Todo applications currently on the market. Identify their strengths and weaknesses and see how you can differentiate your application. Look for opportunities to improve existing solutions.

4. Feature list: Based on user research and competitor analysis, create a list of features you want to include in your Todo app. Prioritize features based on their importance and feasibility.

5. Design and User Experience (UX): Plan the user interface and user experience of the application. Create wireframes or mock-ups to visualize your app's layout and workflow. Ensure an intuitive, user-friendly, and visually appealing design.

6. Technology stack: Decide on the technology stack for your Todo application. Choose the programming language, framework, and database that best meets your project requirements.

7. Data Model: Define the data model for your application. Define how tasks will be stored, including their properties such as title, description, due date, and status (completed or not).

8. Backend and API Design: Backend architecture planning and API design. Decide how you will handle creating, updating, and deleting tasks. Configure the API endpoint to interact with the user interface.

9. Security and authentication: Implement security measures to protect user data and prevent unauthorized access. Plan how users will securely sign up, log in, and manage their accounts.

10. Database Design: Design the database schema to store task data efficiently. Select the appropriate database system (for example: SQL, NoSQL) for your needs.

11. Task management: Plan how users will add, edit, perform, and delete tasks. Consider adding features like task prioritization, categorization, and tags for better organization.

12. Notice: Plan how users will be notified of upcoming or overdue tasks. Consider integrating email notifications or push notifications for mobile apps.

13. Cross-platform compatibility: If you're planning to build a mobile app or a web app, decide which platforms and devices you'll support. Ensure cross-platform compatibility for a wider user base.

14. Testing and Quality Assurance: Plan a test strategy to identify and fix bugs and ensure the stability and performance of the application. Perform manual and automated tests.

15. Deployment and Maintenance: Decide on your application deployment and hosting strategy. Plan how you will handle application updates, bug fixes, and ongoing maintenance.

16. Feedback and Iteration: Once your Todo app is live, collect user feedback and iterate based on their suggestions and needs. Continually improve the app to provide a better user experience.

3. Choice of Framework and Technologies

About HTML

HTML (Hypertext Markup Language) is a standard markup language used to create and structure websites and web applications. It provides a way to define the structure and content of a web page using a set of elements and tags, allowing the browser to interpret and display the content to the user.

Here are some key points about HTML:

1. Markup language: HTML is a markup language, not a programming language. It uses a system of tags (also called elements) to define the structure and content of the web page. Tags are written with curly braces, such as ''.

2. Profile structure: An HTML document usually begins with the declaration '<!DOCTYPE>' followed by '<html>' element that contains the entire content of the web page. '<head>' provides meta information about the document, while '<body>' contains the display content displayed in the browser.

3. Elements and tags: HTML elements represent the building blocks of a web page. Each element is represented by a start tag ('') and an end tag ('') or, in some cases, a self-closing tag (''). The content of an element is placed between the start tag and the end tag.

4. Properties: HTML elements can have attributes that provide additional information about the element. Attributes are placed in the hashtag and provide information such as styling, alignment, or other functionality. For example, 'Link' uses the 'href' attribute to specify the URL the link points to.

5. Semantics: HTML consists of a set of semantic elements that give meaning to the content they contain. For example, '', '', '', '' and '' provide structural context, making it easier for search engines and assistive technologies to understand the content of a web page.

6. Document preview: The combination of semantic and heading elements ('', '', etc.) form the outline of the document. This helps with accessibility and SEO (Search Engine Optimization).

7. Hyperlinks: HTML allows you to create hyperlinks to link one page to another or to specific sections of the same page using the '' element.

8. List: HTML provides ordered ('') and unordered ('') lists, as well as definition lists ('').

9. Form: HTML includes form elements like '<form>', '<input>', '<select>', '<text area>', etc., which allow the user to enter and submit data to the machine owner.

10 Multimedia: HTML supports embedding multimedia content, such as images (''), >'), >'),

Python Flask

Flask is a popular and lightweight web framework for Python, designed to make it easy to build web applications and APIs. It is classified as a micro-framework because it provides only the essentials to get started, keeping the core simple and extensible. Flask is widely used due to its simplicity, flexibility, and ease of use, making it a great choice for both new and experienced developers.

Here are some key features and characteristics of Python Flask:

1. Micro framework: Flask is a micro-framework, which means it provides a minimal set of tools and libraries needed to build web applications. This allows developers to select and integrate additional components as needed, making Flask highly customizable.

2. Minimalism: Flask aims to keep things simple and straightforward. Its core is minimal and focused on the essentials of web development, such as routing, HTTP handling, and template rendering. As a result, Flask has a small and easy to understand code base.

3. Route: Flask uses a decorator-based approach to define routes. Developers can use Python decorators to map URLs to Python functions, making it easy to manage various HTTP methods (GET, POST, etc.), and define logic for each route.

4. Model building: Flask provides a built-in templating engine (Jinja2) that allows developers to separate application logic from presentation. This allows creating dynamic HTML templates with data from Python variables.

5. Integration with other libraries: Flask can be easily integrated with other Python libraries and extensions, providing additional features like database integration, authentication, and more. These extensions are called Flask extensions.

6. Werkzeug and Jinja2: Flask is built on two powerful libraries: Werkzeug (WSGI utility library) and Jinja2 (template engine). These libraries handle many low-level web development tasks, allowing Flask to focus on the application layer.

7. RESTful Support: Flask has built-in support for building RESTful APIs, making it a great choice for building web services and APIs.

8. Widely used and well documented: Flask has a large and active community, which means plenty of documentation, tutorials, and online support. There are many Flask extensions and third-party packages to enhance the functionality.

9. Light and fast: Due to its minimalist nature, Flask is very light and fast. It is suitable for small and medium sized applications and APIs.

10. Open-source code: Flask is an open-source project released under the BSD license, allowing free use and distribution.

Overall, Python Flask is a great web framework for developers who want a simple yet powerful solution for building web applications and APIs. Its flexibility, ease of use, and strong community support have made it a popular choice for many web developers around the world.

4.Flow Diagram

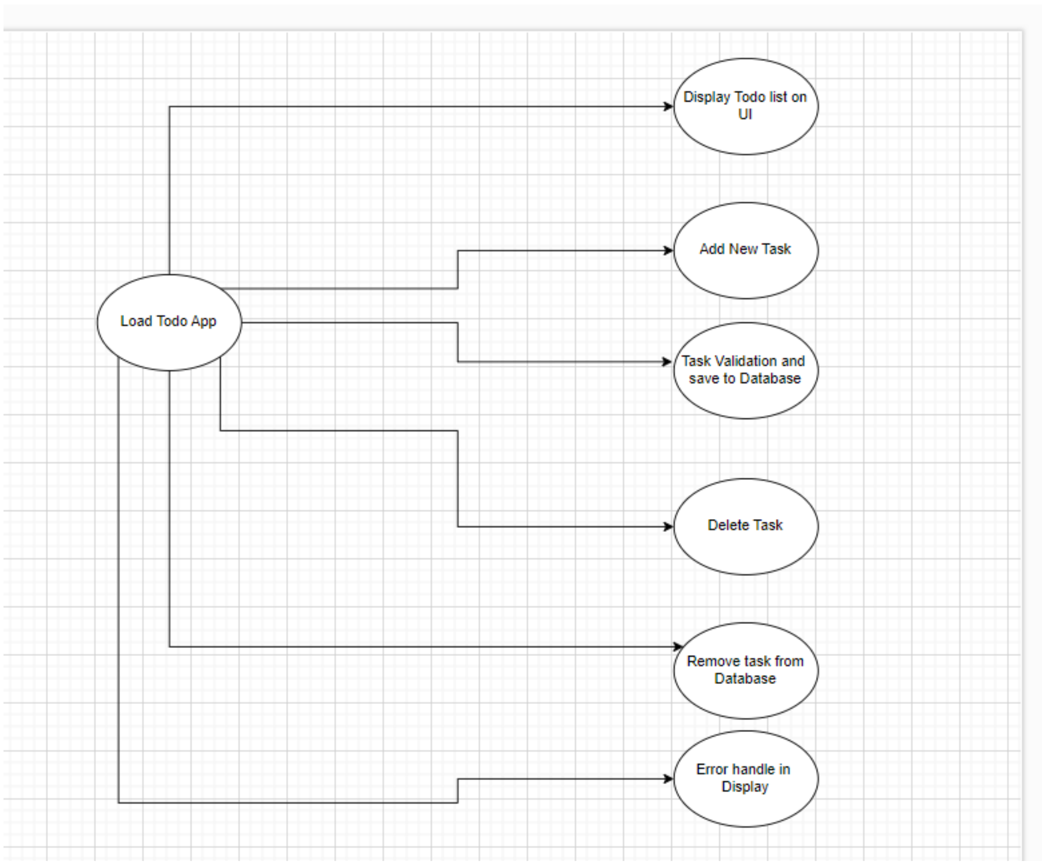


Figure 1. Flow diagram of Todo list application

4.1 Flow-Diagram explanation

Download the Todo app: The process starts when the user opens the Todo app.

Show the task list on the UI: The application retrieves the list of available tasks from the database and displays them on the user interface.

Add a new task: The user can enter a new task in the "Add New Task" input field.

Validate the task and save to the database: The application validates the input task, making sure it's not empty or invalid. If the task is valid, it will save it to the database.

Edit an existing task: The user can choose to edit an existing task by clicking the "Edit" button associated with the task.

Update task in database: The application updates the task in the database with the new changes made by the user.

Mark the task as complete: Users can mark a task as completed by clicking the "Check" button associated with that task.

Mark the task as incomplete: The user can cancel the task completion by clicking the "Uncheck" button associated with the task.

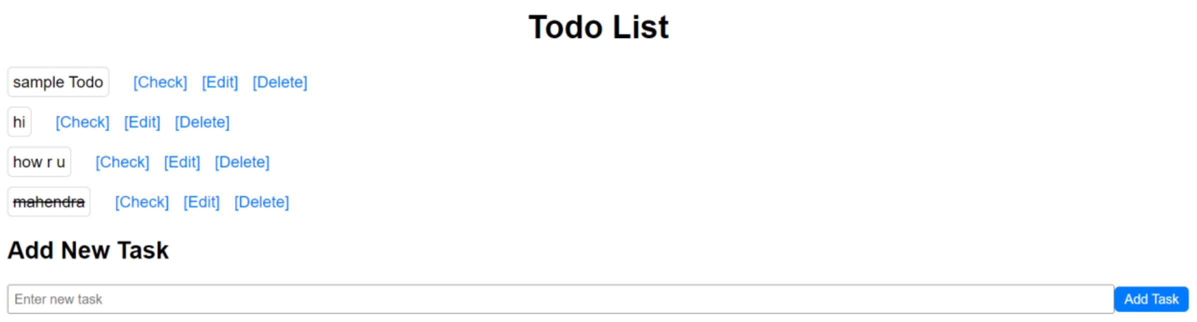
Delete task: A user can delete a task by clicking the "Delete" button associated with the task.

Remove the task from the database: The application deletes the task from the database after the user deletes the task.

Error handling and display: The application handles errors skilfully, such as invalid entries, database connection problems, etc. and display the appropriate error messages to the user.

This flowchart provides a high-level overview of how users interact with the Todo application and how the application manages these interactions by updating the database and displaying relevant information on the user interface. Actual implementation details may vary depending on specific requirements and Todo application design.

5.Home page


Todo List

sample Todo [Check] [Edit] [Delete]

hi [Check] [Edit] [Delete]

how r u [Check] [Edit] [Delete]

~~mahendra~~ [Check] [Edit] [Delete]

Add New Task

Enter new task

Figure 2. Home page of Todo list application

The content on the main page of the Todo app is a central area where users can interact with their tasks and manage their to-do lists efficiently. It typically includes various elements and components that provide an overview of tasks, allow users to add new tasks, edit existing tasks, mark tasks as completed, and perform other actions. other task-related actions. Here are the main components commonly found on the main page of the Todo app:

5.1 Home page Features:

1. To-do list view: This section displays the task list. Each task is displayed as a separate item with a title, description, due date, and status (completed or pending). Completed tasks can have visual indicators to distinguish them from pending tasks, such as checkboxes or line effects.

2. Action Mission: Each action item can have actions or buttons to allow the user to perform different actions, such as editing the task, marking the task as completed or pending, and deleting the task. there.

3. Add new quest item: A prominent input field is provided so that users can enter a new task that they want to add to their task list. This input field can also allow the user to set a due date or select a task category.

4. Add action button: Next to add task, there is a button (for example, "Add task" or "Create") that allows the user to submit a new task.

5. Category or Task Card: Users can have the option to categorize tasks by different tags or categories. This allows them to better organize their tasks and easily find related tasks.

6. Sort and Filter Tasks: The main page can include options to sort tasks by different criteria, such as due date, priority, or completion status. Users can also have the option to filter tasks based on specific categories or tags.

7. Number of quests and stats: This page can show the total number of tasks as well as the number of completed and pending tasks for quick reference. It can also show statistics like the number of tasks completed today or this week.

8. Search function: The main page may include a search bar to allow users to search for specific tasks based on keywords or other filters.

9. User account information: If the application requires user authentication, the main page may display the user's account information, such as their username or profile picture.

10. Navigation Bar or Sidebar: The main page can have a navigation bar or a sidebar for accessing other parts of the Todo app, such as settings, completed tasks, or task categories.

11. User-friendly design: The main page is designed to be intuitive and user-friendly, allowing users to easily navigate, interact, and manage their tasks.

12. Responsive Design: The main page is typically designed to be responsive, adapting to different screen sizes and devices, including desktops, tablets, and mobile phones.

Overall, the content on the main page of the Todo app provides users with a clear overview of their tasks and facilitates efficient task management. The layout, design, and functionality of the main page have a significant impact on the overall user experience and usability of the app.

As an AI language model, I don't have access to external sources such as specific Todo Apps for generating Harvard references. However, I can guide you on how to create a Harvard reference for a Todo App if you have used a specific website, article, or academic paper as a reference for your app's development.

6. Add Task

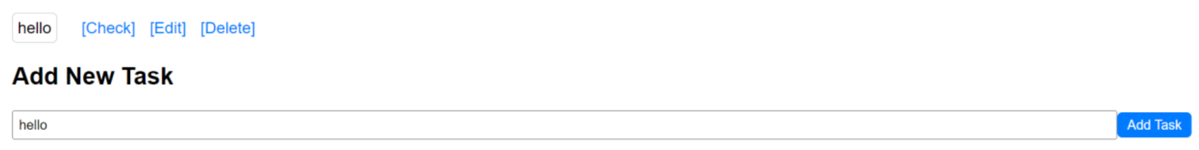


Figure 3. Add Task functionality.

In the above figure we can add task, if you add task that was appear in the top. In this figure I was written hello, that can appear in the screen.

7. Remove Task

Figure 4. Remove task functionality.

In this above figure we can remove the task, if you remove the task the appear code is removing entire screen. That place is shown in empty.

8. Testing and Quality Assurance Todo app

Testing and quality assurance are essential aspects of the Todo application development process. Proper testing ensures that the application works as expected, without fatal errors, and meets user requirements. Here are the main steps involved in testing and QA of the Todo app:

1. Unit test: Execute unit tests to test individual application components. Write test cases for functions and methods that perform specific tasks, such as adding a task, marking it as complete or deleting it. - Use testing frameworks like 'unittest' or 'pytest' in Python to automate the testing process.

2. Integration testing: Perform integration tests to verify that the different components of the application work properly with each other. Check front-end and back-end interactions to ensure data is transmitted correctly.

3. User Acceptance Testing (UAT): Engage end users or beta testers to do user acceptance testing. Collect user feedback on the app's usability, look, and functionality.

4. Test function: Perform functional testing to verify that the application meets the specified functional requirements. Test all features, such as adding, editing, and deleting tasks, to make sure they work as expected.

5. Usability test: Evaluate the usability of the application through usability testing. Make sure the app is intuitive and easy to use for the target audience.

6. Check compatibility: Test the app on different browsers (e.g. Chrome, Firefox, Safari) and devices (e.g. desktop, tablet, mobile) to ensure compatibility.

7. Performance test: Perform performance tests to evaluate application response time and resource usage under various conditions. Check for possible bottlenecks and optimize application performance if needed.

8. Security check: Perform security testing to identify and address potential security vulnerabilities. - Validate input data to avoid common security issues like SQL injection and cross-site scripting (XSS).

9. Error and exception handling: Check the application's error and exception handling to make sure the application properly handles unexpected situations. Provide meaningful error messages to users in case of failure.

10. Regression testing: Perform regression testing after each code update or change to ensure that new features don't cause new problems or break existing functionality.

11. Check accessibility: Test the app's accessibility for users with disabilities, ensuring the app meets accessibility standards.

12. Try to load (optional): If the application is expected to handle many users, consider load testing to evaluate the application's performance under heavy load.

13. User feedback and iteration: Collect feedback from users and beta testers throughout the testing process. Troubleshoot and iterate on the app based on user feedback.

14. Documents: Record test cases, test procedures, and test results for future reference and collaboration. By thoroughly testing and implementing quality assurance, developers can ensure that the Todo app is reliable, user-friendly, and meets the desired standards and requirements.

9. Conclusion

Overall, Todo app development is a valuable learning experience for the development team. He provided information on web application development, user experience design, and the importance of testing and quality assurance in delivering a successful product.

As the app continues to grow and thrive, we remain committed to maintaining and improving the app's functionality and usability, enriching the user experience. We express our gratitude to all stakeholders, users and contributors who have supported the development and success of the project. With the Todo app, we aim to make task management a seamless and rewarding experience for users, allowing them to easily accomplish their goals and tasks.

10. Reference

Author(s) Last name, Initials. (Year). Title of webpage/document. Website Name. [Online]

Available at: URL [Accessed Date].

Here's a hypothetical example of a Harvard reference for a website that provides information about building a Todo App:

Smith, J. (2023). How to Build a Todo App: A Comprehensive Guide. TodoAppTutorials. [Online]

Available at: https://www.todoapptutorials.com/build-todo-app [Accessed 1st August 2023].

Please note that the above example is fictional and only serves as an illustration. For creating accurate Harvard references, always use the specific details of the source you are referencing, including the actual author's name, publication date, title, and URL.

You tube link: https://www.youtube.com/watch?v=W1r8fVLS-gI&ab_channel=NeuralNine