9530

St. MOTHER THERESA ENGINEERING COLLEGE

COMPUTER SCIENCE ENGINEERING

NM-IDNM-ID: 4B28EF483495A768DF045DCC0A2CFD0E

REG NO: 953023104097

DATE: 22-09-2025

Completed the project named as

Phase III

FRONTEND TECHNOLOGY

TO DO LIST APPLICATION

SUBMITTED BY:

REGINA BEULA.P

8754266260

Project Setup

The Project Setup stage is one of the most critical steps in the development of the To-Do List Application, as it establishes the foundation upon which the entire project will be built. A well-organized setup not only ensures smooth development but also improves collaboration, scalability, and maintainability throughout the project lifecycle.

The process begins with technology stack selection, which defines the tools and frameworks to be used in the project. For the frontend, frameworks such as React.js, Angular, or Vue.js are commonly chosen because they provide responsive, dynamic, and user-friendly interfaces. Since the To-Do List application heavily relies on user interaction, a frontend library like React.js is ideal due to its component-based architecture and strong community support. On the backend, Node.js with Express.js can be employed to provide RESTful APIs for task management, user authentication, and data synchronization. For storing tasks and user data, databases like MongoDB (NoSQL) or MySQL/PostgreSQL (relational) can be used depending on whether the application requires flexible schema handling or strict relational design.

After finalizing the stack, the development environment is prepared. This involves installing essential software such as Node.js runtime, npm (Node Package Manager), Visual Studio Code (IDE), and database servers. Developers also configure extensions, linters, and code formatters (like ESLint or Prettier) to maintain clean and standardized code across the project. To further streamline the setup, a package.json file is created to handle dependencies, making it easy for any team member to replicate the environment using a single command.

Next, project initialization is carried out. A new Git repository is created and connected to GitHub or GitLab for version control and collaboration. Git enables tracking of changes, branch management for different features, and rollback in case of issues. Branching strategies such as Git Flow or feature-based branching ensure organized development. Additionally, configuration files such as .gitignore are set up to exclude unnecessary files from version control, while .env files are prepared to securely store environment variables like database credentials or API keys.

The project folder structure is then designed carefully to promote scalability and ease of maintenance. The frontend may contain folders such as components, pages, services, and assets, while the backend may include routes, controllers, models, and middlewares. This separation of concerns ensures that each part of the application is modular and easier to debug or extend in the future.

Documentation is also an integral part of project setup. A README.md file is created to guide developers with instructions on installing dependencies, running the development server, and contributing to the project. For larger teams, contribution guidelines and coding standards are documented to maintain consistency.

Finally, the project setup also integrates testing and deployment preparation. Testing frameworks such as Jest or Mocha are configured to ensure features can be validated from the start. Continuous integration tools (like GitHub Actions) may also be linked to automate testing and deployment.

By the end of this stage, the To-Do List Application has a well-structured, collaborative, and ready-to-code environment. This ensures that development progresses efficiently, errors are minimized, and the foundation is strong enough to support all subsequent phases of implementation.

Core Features Implementation

The Core Features Implementation phase represents the heart of the To-Do List Application development process. It is during this stage that the essential functionalities, which form the minimum viable product (MVP), are translated from design into working software. The primary goal is to ensure that the application enables users to manage tasks seamlessly, while also providing a foundation upon which advanced features can later be built.

The first and most critical feature to implement is Task Management. This involves allowing users to create, read, update, and delete (CRUD) tasks. For instance, when a user adds a new task, the frontend captures the input through a form and communicates with the backend API to save the data in the database. Each task typically includes fields such as title, description, due date, and completion status. On the user interface, tasks are displayed in an organized list, where users can easily check or uncheck completed tasks. The backend logic ensures that changes are instantly updated in the database, maintaining synchronization between client and server.

The next major feature is Task Categorization and Prioritization. Users should be able to categorize tasks into groups such as work, personal, or study. Additionally, tasks can be prioritized as high, medium, or low, helping users focus on urgent activities. Implementation of this feature requires designing appropriate database schemas to handle categories and priority levels, along with UI elements such as dropdowns or color-coded labels for easier visualization.

User Authentication and Authorization is another crucial component. Since the To-Do List application may cater to multiple users, each user must have a secure account where their personal tasks are stored. This feature is implemented using authentication mechanisms such as

JWT (JSON Web Tokens) or OAuth, which ensure secure login and session management. Passwords are hashed before being stored in the database, and APIs are protected so that users can only access their own tasks.

Another core implementation is Task Filtering and Search. As users accumulate multiple tasks, they should be able to search for specific items or filter based on categories, priorities, or due dates. This feature involves both frontend logic for filtering displayed items dynamically and backend query optimization to fetch relevant results efficiently.

Task Status Tracking is also fundamental. Users must be able to mark tasks as complete or incomplete with a simple click. This requires updates to the database and real-time UI feedback, often implemented with asynchronous calls (using Axios or Fetch API). Additionally, visual indicators such as checkmarks or strikethroughs are provided to enhance the user experience.

To ensure data is not lost, Data Persistence is implemented either through a local state (for offline functionality) or a connected database (for online storage). Technologies such as MongoDB or MySQL ensure that tasks remain available even after users close the application or switch devices.

Finally, Testing and Debugging of core features are performed to validate functionality. Unit tests confirm that each feature works independently, while integration tests check that the frontend, backend, and database communicate correctly.

Data Storage (Local State / Database)

Data storage plays a critical role in the functionality of a To-Do List Application, as it ensures that user-created tasks are preserved and retrievable. Without an effective storage mechanism, users would lose their data after each session, making the application unreliable. Therefore, implementing a proper data storage strategy using local state management and a database is essential.

At the frontend level, local state is often used to temporarily hold data while the application is running. For instance, when a user creates a new task, it is first stored in the application's state using frameworks such as React's useState/useReducer or Angular's services. This approach ensures real-time updates, so tasks instantly appear in the user interface without requiring a page refresh. However, since local state is temporary, the data is lost once the application is closed or refreshed, which is why it must be combined with database storage.

On the backend, a database provides persistent storage, enabling tasks to remain available across multiple sessions and devices. For example, each task can be stored with attributes such as task ID, title, description, deadline, status, and category. Databases like MongoDB (NoSQL) or

MySQL/PostgreSQL (SQL) are commonly used. MongoDB offers flexibility with unstructured data, while SQL databases provide strong relational integrity for structured task management.

The connection between local state and database is managed through API calls. When users add, update, or delete tasks, the frontend sends requests to the backend, which processes the data and updates the database. The updated data is then reflected in the local state to maintain consistency.

Testing Core Features

Testing the core features of the To-Do List Application is a crucial step in ensuring that the system functions smoothly, delivers accurate results, and provides a reliable user experience. The primary objective of testing is to validate whether the implemented functionalities meet the requirements defined during the design and planning phases.

The first layer involves unit testing, where each feature is tested in isolation. For instance, adding a task, editing a task, or marking it as complete is tested individually to ensure that the functions perform correctly. Unit testing also helps developers quickly identify and fix bugs before they affect the entire system.

Following this, integration testing is performed to check whether different modules interact seamlessly. For example, when a new task is added on the frontend, it should correctly communicate with the backend API and be stored in the database. Similarly, any update in the database should be reflected in the user interface without delays. Integration testing ensures the consistency of data across components.

User interface (UI) testing is also vital, as the application's usability directly affects user satisfaction. This testing ensures that buttons, forms, task lists, and filters respond correctly to user interactions. Automated tools like Cypress or Selenium can simulate real user actions to confirm that the app behaves as expected.

In addition, performance testing evaluates how well the application handles multiple users and a growing number of tasks, ensuring there are no crashes or lags. Security testing verifies that sensitive information, such as user credentials, is protected and that only authenticated users can access their data.

Finally, user acceptance testing (UAT) involves real users evaluating the application to confirm that it meets expectations. This step ensures the To-Do List Application is practical, reliable, and ready for deployment.

Step-by-Step Guide for Using GitHub with the To-Do List Application

1. Install Git

- Before starting, ensure Git is installed on your system. Check by running:

```git --version```

- If not installed, download from https://git-scm.com/.

2. Initialize a Local Repository

- Navigate to your project folder and initialize Git:

```cd path/to/todo-list-app

git init```

- This creates a `.git` folder to start tracking changes.

**3. Configure Git (Optional but Recommended)**

Set your name and email for commit tracking:

```

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

```

**4. Check Repository Status**

- See which files are untracked or modified:

```

git status

```

**5. Add Files to the Staging Area**

- Prepare files for the first commit:

```
git add .
```

(`.` adds all files; you can also specify individual files.)

6. Commit Changes

- Save a snapshot of the current state:

```
git commit -m "Initial commit: Setup project structure"
```

7. Create a Repository on GitHub

I. Log in to GitHub.

II. Click **New repository**.

III. Name it `todo-list-app` (or your choice).

IV. Choose visibility: Public or Private.

V. Click **Create repository**.

8. Link Local Repository to GitHub

- Copy the GitHub repository URL and run:

```
git remote add origin https://github.com/username/todo-list-app.git
```

9. Push Local Code to GitHub

- Send your local commits to the remote repository:

```
git push -u origin main
```

> If the default branch is `master`, replace `main` with `master`.

10. Create and Work on Branches

- For adding a new feature (e.g., task categorization):

```
git checkout -b feature/task-categorization
```

- Make changes, stage, and commit:

```
git add .
git commit -m "Add task categorization feature"
```

- Then merge it back to main after review:

```
git checkout main
git merge feature/task-categorization
```

11. Pull Updates from GitHub

- Keep your local repository updated with team changes:

```

git pull origin main

```

12. Clone an Existing Repository

- If starting from someone else's repo:

```

git clone https://github.com/username/todo-list-app.git

cd todo-list-app

```

13. Optional: View Commit History

- Check all commits:

```

git log

```

- Show a compact history:

```

git log --oneline

```

14. Using Pull Requests (PR)

1. Push your feature branch to GitHub:

```
```

git push origin feature/task-categorization

```

2. Go to GitHub → Open a pull request from your branch to main.

3. Team members review code, suggest changes, then merge.

**15. Best Practices**

- Commit frequently with clear messages.
- Use branches for every new feature or bug fix.
- Regularly pull changes from `main` to avoid conflicts.
- Document issues and tasks using GitHub Issues and Project Boards.
```