COLLEGE CODE: 9528

COLLEGE NAME: SCAD COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

STUDENT NM ID: 172747DBD1D6C799686466A00132E028

Roll no : 952823104116

DATE : 26.09.2025

Completed the project named as:

Phase 2

TECHNOLOGY PROJECT NAME :

TO DO LIST APPLICATION

SUBMITTED By,
NAME: NAMA KANI A
MOBILE NO:9944011925

# Phase 2 – Solution Design & Architechture

## 1. Tech Stack Selection

For the frontend, the application uses **React.js** for the web platform and **Flutter** for mobile development. React.js provides a component-based architecture, fast rendering, and an intuitive user interface, while Flutter allows cross-platform mobile development with a single codebase and high-performance UI. Together, they ensure a seamless, responsive, and user-friendly experience across devices.

The backend is built using **Node.js with Express**, which offers a lightweight, scalable, and efficient way to handle APIs and multiple concurrent requests. For data management, **PostgreSQL** is chosen as the relational database due to its reliability and structured data handling, while **Redis** is integrated for caching to improve response times and

performance. Authentication is handled using **JWT**, with optional **OAuth2** support for social logins, ensuring secure and flexible user access.

## 2. UI Structure / API Schema Design

The **UI Structure** defines the overall layout, navigation flow, and major screens of the application. It ensures users can easily interact with the system to perform core functions such as creating, viewing, updating, and managing tasks.

### Key Screens

1. **Login / Signup Page**
    - Entry point of the app where users can register or log in securely.
2. **Dashboard / Home**
    - Central workspace showing all tasks, with filters for categories, status, or deadlines.
3. **Task Management Page**
    - Screen to add, update, delete, or mark tasks as completed.

4. **Calendar View**
    - Displays tasks by date, helping users plan ahead visually.
5. **Notifications / Reminders**
    - Lists alerts and reminders about pending or overdue tasks.
6. **User Profile / Settings**
    - Contains user information, preferences, and logout option.

## API Scheme Design

The **API Schema Design** defines how the frontend communicates with the backend using RESTful endpoints. It specifies the resources (like users, tasks, categories) and the operations (CRUD – Create, Read, Update, Delete) that can be performed on them. A well-designed schema ensures **scalability, reusability, and security** of the system.

## Authentication

`POST /api/auth/signup` → Register a new user
`POST /api/auth/login` → Authenticate user & return JWT
`POST /api/auth/logout` → End user session

## Tasks

`GET /api/tasks` → Retrieve all tasks of a user
`GET /api/tasks/:id` → Retrieve specific task details
`POST /api/tasks` → Create a new task
`PUT /api/tasks/:id` → Update an existing task
`DELETE /api/tasks/:id` → Remove a task

## Categories / Tags

`GET /api/categories` → Fetch all categories for user
`POST /api/categories` → Create a new category
`DELETE /api/categories/:id` → Delete a category

## Notifications

`GET /api/notifications` → Fetch task reminders/alerts

## Data Handling Approach

The To-Do List application stores structured information such as users, tasks, categories, and reminders in a **relational database (PostgreSQL)**, ensuring data consistency and reliability. To enhance performance, frequently accessed data such as task lists and user sessions is cached using **Redis**, reducing query response times. Regular automated backups and cloud-managed services provide high availability and disaster recovery, ensuring no critical information is lost.

Data is managed through **RESTful APIs** that handle all CRUD (Create, Read, Update, Delete) operations on tasks and users. An **ORM (Object-Relational Mapping)** tool like Sequelize is used to simplify database interactions and maintain schema consistency. Advanced filtering and search functions are supported by indexed queries, while **WebSockets** provide real-time synchronization of tasks across multiple devices. This ensures smooth performance and up-to-date task information for users.

## 3. Component / Module Diagram

A **Component or Module Diagram** represents the logical structure of the system by breaking it down into smaller, manageable units (modules). Each module is responsible for a specific functionality, and together they form a complete system. This ensures **modularity, scalability, and easier maintenance**.

### Core Modules / Components

4. **Authentication Module**
    a. Handles user signup, login, logout, password management, and session handling.
5. **Task Management Module**
    a. Manages CRUD operations for tasks (create, read, update, delete).
    b. Supports priorities, deadlines, and task status (pending, completed).
6. **Category & Tag Module**
    a. Organizes tasks under categories/tags for better grouping and filtering.

7. **Notification & Reminder Module**
    a. Sends push/email notifications for due tasks and reminders.
8. **User Profile & Settings Module**
    a. Manages user preferences, profile updates, and app settings.
9. **Database Module**
    a. Stores and retrieves data (users, tasks, categories, notifications).
10. **API/Integration Module**
    a. Provides REST APIs for communication between frontend and backen

**Basic Flow Diagram**

A **Basic Flow Diagram** represents the high-level flow of how users interact with the application and how data moves between the frontend, backend, and database. It focuses on the main processes: authentication, task management, and notifications.

**Flow Steps**

1. **User Access**
    - User opens the app (Web/Mobile).

- Proceeds to **Login / Signup**.

2. **Authentication**
   - User submits credentials → **Auth Module** verifies via backend.
   - On success → JWT token issued → user enters dashboard.
3. **Task Management**
   - User performs CRUD operations on tasks (create, view, update, delete).
   - Requests sent via **API** to backend → processed and stored in **Database**.
4. **Categories & Tags**
   - User organizes tasks into categories/tags.
   - Data stored and retrieved from the database.
5. **Notifications / Reminders**
   - Backend checks due dates using scheduler.
   - Sends reminders via **Firebase (push)** or **SendGrid (email)**.
6. **User Profile & Settings**
   - User updates profile or preferences.
   - Changes saved to database and reflected in frontend.