College Code : 9509
College Name : Holy Cross Engineering College
Department : Computer Science Engineering
Student NM id DAE074EB4A8886B8B224E01DC0C33A55
Roll No : 950923104044
Date : 15.09.2025
Project Name : IBM-FE-List-to do application

Completed the project named as

Sumbitted By,
Sanjay K
9342667280

Phase 2 — Solution Design & Architecture

(Deadline – Week 7)

This document presents the solution design and architecture for a To-Do List Application. It covers tech stack selection, UI structure, API schema, data handling, component diagrams, flowcharts, and an example program to implement the backend.

1. Tech Stack Selection

• Frontend: React.js + Tailwind CSS
• Backend: Node.js + Express.js
• Database: MongoDB (Atlas)
• Authentication: JWT
• Hosting: Vercel/Netlify (Frontend), Render/Heroku (Backend)

2. UI Structure

• Login / Signup Page
• Dashboard (Task List)
• Task Input Form (Title, Description, Deadline, Priority)
• Task Item (Edit / Delete / Complete)
• Filter & Sort

3. API Schema Design

Auth APIs:
- POST /auth/signup → Register user
- POST /auth/login → Login and return JWT

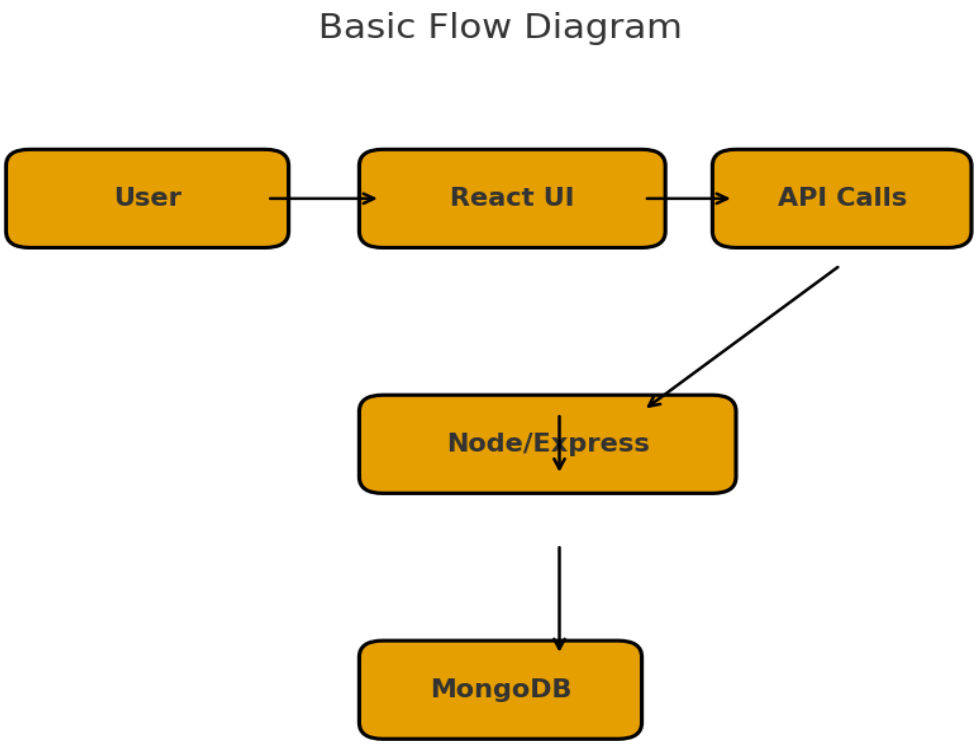
Task APIs:
- GET /tasks → Fetch all tasks for a user
- POST /tasks → Create a new task
- PUT /tasks/:id → Update task
- PATCH /tasks/:id/complete → Mark task as completed
- DELETE /tasks/:id → Delete task

Example Backend (Node.js + Express) - Minimal Server

```
// server.js const express = require('express'); const mongoose = require('mongoose');
const bodyParser = require('body-parser'); const jwt = require('jsonwebtoken'); const app
= express(); app.use(bodyParser.json()); // Mongoose models (simplified) const taskSchema
= new mongoose.Schema({ userId: mongoose.Types.ObjectId, title: String, description:
String, deadline: Date, priority: String, status: { type: String, default: 'Pending' },
createdAt: { type: Date, default: Date.now } }); const Task = mongoose.model('Task',
taskSchema); // Routes (simplified) app.post('/tasks', async (req, res) => { const task =
new Task(req.body); await task.save(); res.status(201).json(task); }); app.get('/tasks',
async (req, res) => { const tasks = await Task.find({ userId: req.query.userId });
res.json(tasks); }); // Start (replace with your MongoDB URI)
mongoose.connect(process.env.MONGO_URI || 'mongodb://localhost:27017/todo_app') .then(()
=> app.listen(3000, () => console.log('Server running on port 3000'))) .catch(err =>
console.error(err));
```

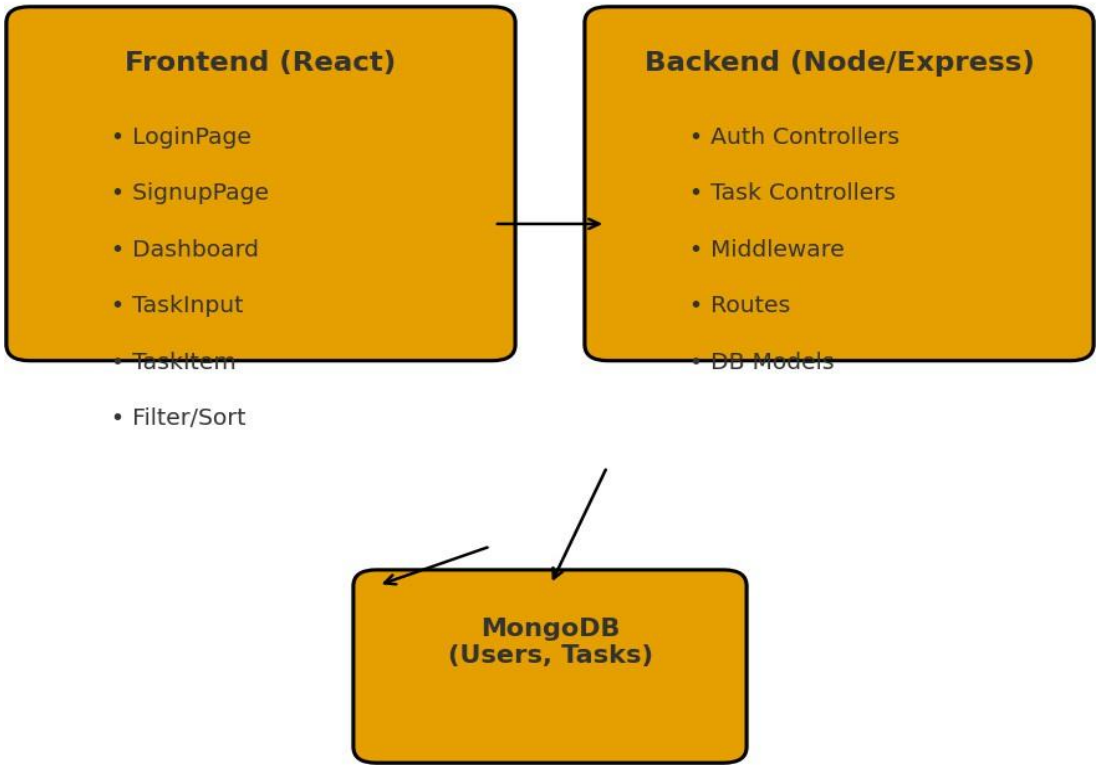
4. Diagrams

Basic Flow Diagram



Component / Module Diagram

Component / Module Diagram



5. Data Handling Approach

• Frontend: React hooks (useState, useEffect) or Context for global state; Axios for API calls.
• Backend: Express middleware for validation (e.g., Joi), JWT for auth, structured controllers.
• Database: MongoDB with Mongoose models for Users and Tasks.
• Security: Store JWT in HTTP-only cookies, validate inputs, and use rate limiting in production.

This 5-page document provides a concise, presentation-ready overview of Phase 2 for the To-Do List application, including diagrams and an example backend program to get started.