

NOTEBOOK: THE TO-DO LIST APPLICATION

CSCE5430.001 Software Engineering

By:

Gayathri Baman
University of North Texas
Master's in Computer Science

ABSTRACT:

A Notebook is a simple yet powerful tool used for task management and organization. It typically includes a list of tasks or items that need to be completed, often with deadlines or priorities attached to each item. The primary purpose of a Notebook is to serve as a reminder and a guide, helping individuals or teams keep track of their responsibilities and goals. By breaking down larger projects into smaller, manageable tasks, Notebook can increase productivity and reduce the feeling of being overwhelmed. They can be created in various formats, such as paper lists, digital apps, or even mental notes, and can be tailored to fit personal or professional needs. Notebooks often evolve throughout a day or project, with items being added, modified, or checked off upon completion. This tool is widely used in both personal and professional contexts to enhance efficiency and organization.

The development of this application involves following key-phases:

Requirement Analysis: Understand client needs clearly.

Design & Architecture: Plan a scalable framework.

Coding Practices: Write clean, efficient code.

Testing: Ensure reliability via unit, integration, system tests.

Maintenance & Updates: Adapt to changing needs.

Project Management: Coordinate for successful delivery. It guides software teams for quality, efficiency, and client satisfaction.

Table of content:

1. Introduction.....	5
2. Purpose.....	6
3. Overall Description	7
3.1 Production Functions	7
3.2 Use Case Model Survey	7
4. Requirement Analysis	9
4.1 System Design	10
4.1.1 Context Diagram	10
4.1.2 Class Diagram	12
4.1.3 System Architecture Diagram	14
4.1.4. Activity Diagram	16
4.1.5 Data Flow Diagram	18
5. Specific Requirement	18
5.1 Hardware requirement	18
5.2 Software Requirement	18
6. Implementation	18
6.1 Project Overview	18
6.2 Technology Stack.....	19
6.3 Key Features.....	19
6.4 Development Approach	19
6.5 Challenges Encountered	19
6.6 Code	20
6.7 Results	32
7. Testing	36
8. Deployment	39
9. Conclusion	42

Table of Figures:

S. No	Name	Page Number
1.	Use-Case Diagram	10
2.	Context Diagram	11
3.	Class Diagram	13
4.	System Architecture Diagram	15
5.	Activity Diagram	17
6.	Data Flow Diagram	18
7.	Deployment Diagram	41

1.Introduction:

Software engineering, characterized by its systematic, disciplined, and quantifiable approach, demands meticulous planning and organization. In this context, a Notebook is not merely a set of tasks but a fundamental framework that encapsulates the lifecycle of software development. This comprehensive Notebook serves as a pivotal guide, facilitating software engineers and project managers in navigating through the intricate phases of software creation and implementation.

Bridging Vision with Reality: At the core of any software project lies a vision - a solution to a problem or a means to fulfill a specific need. The Notebook begins by translating this vision into actionable steps. It starts with requirement gathering and analysis, ensuring that the software solution is perfectly aligned with the client's needs and expectations.

Design and Architecture Planning: Once the requirements are clear, the next set of tasks revolves around designing the software's architecture. This involves making crucial decisions on the tech stack, database design, and overall system architecture. The Notebook serves as a guide to ensure that the design is scalable, robust, and efficient.

Coding and Development: The most extensive section of the Notebook relates to coding and development. This section delineates tasks related to writing clean, efficient, and secure code, implementing features, and adhering to coding standards. It emphasizes the importance of version control, code reviews, and maintaining documentation, which are vital for the long-term sustainability of the software.

Testing and Quality Assurance: No software engineering project is complete without rigorous testing. The Notebook encompasses various forms of testing - unit, integration, system, and user acceptance testing. These tasks ensure that the software is reliable, meets quality standards, and is free from bugs or errors.

Deployment and Maintenance: The final phase of the Notebook involves deploying the software to a production environment and its subsequent maintenance. This includes tasks related to deployment strategies, monitoring system performance, and providing updates and fixes.

Project Management and Team Collaboration: Throughout the software development process, effective project management and team collaboration are essential. The Notebook includes tasks related to managing timelines, resource allocation, regular team meetings, and communication strategies to ensure that the project stays on track and team members are synchronized.

This Notebook is designed to be a living document, adaptable to the changing dynamics of software projects. It not only helps in organizing tasks but also serves as a compass, guiding the team through the complexities of software development, ensuring a harmonious blend of technical excellence and project management acumen. The goal is to streamline the process, reduce risks, and lead the project towards a successful and timely completion.

2.Purpose:

The main purpose of implementing a Notebook project is multifaceted and instrumental to the success of the project. Here are the key purposes it serves:

Clarifies Objectives and Priorities: A Notebook helps in clearly defining the objectives of your software project. It outlines the specific tasks that need to be completed, ensuring that every team member understands what is expected. This clarity is crucial for aligning the team's efforts with the project's goals.

Enhances Organization and Planning: Software projects involve numerous tasks, ranging from simple coding assignments to complex system integrations. A Notebook organizes these tasks in a structured manner, making it easier to plan and allocate resources effectively.

Facilitates Task Tracking and Progress Monitoring: With a Notebook, you can track the progress of individual tasks and the project as a whole. This visibility is essential for monitoring project timelines, identifying potential delays early, and making necessary adjustments to stay on schedule.

Improves Productivity and Efficiency: By breaking down the project into smaller, manageable tasks, a Notebook helps in reducing the complexity and overwhelming nature of large projects. It enables team members to focus on one task at a time, which can significantly improve productivity and efficiency.

Promotes Accountability and Responsibility: Assigning specific tasks to team members with clear deadlines fosters a sense of accountability and responsibility. This can lead to a more engaged and motivated team, as each member understands their role and contributes to the project's success.

Aids in Risk Management: A comprehensive Notebook includes not just development tasks but also items related to testing, quality assurance, and contingency planning. This approach is crucial for identifying and mitigating risks early in the project lifecycle.

Ensures Continuity and Knowledge Transfer: In a dynamic project environment, changes in team composition can occur. A detailed Notebook serves as a knowledge base, ensuring that new team members can quickly come up to speed and continue the work without significant disruptions.

Supports Effective Communication: A Notebook acts as a communication tool, providing a common reference point for the entire team. It helps in reducing misunderstandings and ensuring that everyone is on the same page regarding the project's status and next steps.

Encourages Continuous Improvement: By regularly reviewing and updating the Notebook, your team can reflect on the project process, learn from completed tasks, and apply these insights to improve future work.

In summary, this project serves as a crucial tool for ensuring organized, efficient, and effective progression through the project lifecycle, ultimately contributing to the successful delivery of a high-quality software product.

3.Overall Description:

This section of the report will give a general understanding about the project by explaining the factors that affect the product and its requirements.

3.1 Product Functions:

This section serves a higher-level specification of the main functionalities that the system will provide. The product functions are as follows:

1. Database management
2. User authorization
3. Creation of Notes
4. Deletion of Notes
5. Reminder for Notes
6. Subscription for applications

3.2 Use Case Model Survey:

3.2.1 Database management" Use case

Name	Database Management
Description	All the user details and credentials are stored in a text file database for future requirements and authentication
Actors	Actors

3.2.2 User Authorization Use Case

Name	User Authorization
Description	Users get authorized by asking to enter a password in order to purchase products
Actors	Admin

3.2.3 Create Notes Use Case

Name	Create Note
Description	User can create a note of their choice and insert the context as they wish
Actors	User

3.2.4 Deletion of Notes Use case

Name	Deletion of Task or Note
Description	User can delete a note of their choice
Actors	User

3.2.5 Reminder for Notes Use Case

Name	Reminder of Task or Note
Description	User can set reminder for a task or note that they want to be reminded for a particular date and time
Actors	User

3.2.5 Subscription Use Case

Name	Subscription for the application
Description	Users can subscribe to the Notebook application to experience premium features such as insertion of media, adding any number of notes per day without any delay.
Actors	User

4.Requirement Analysis:

A requirement analysis for a Notebook project involves a comprehensive breakdown of the functional and non-functional requirements, user needs, and system specifications. This analysis is crucial for ensuring that the final product meets expectations and solves the intended problems.

Functional Requirements:

User Account Management:

- Users can create, edit, and delete their accounts.
- Authentication mechanisms (login/logout) with security measures.
- Password recovery and user verification processes.

Task Management:

- Ability to add, edit, and delete tasks.
- Set deadlines, priorities, and categorize tasks.
- Option to mark tasks as completed/incomplete.

List Organization:

- Create multiple Tasks for different purposes or projects.
- Ability to sort and filter tasks within lists (e.g., by date, priority).
- Options for customizing the appearance of lists (e.g., color coding).

Notifications and Reminders:

- Push notifications for critical tasks or updates.

Collaboration Features (if applicable):

- Option to share lists or tasks with other users.
- Ability to assign tasks to others and track progress.
- Real-time updates and collaborative editing capabilities.

Data Sync and Accessibility:

- Cross-platform synchronization (if applicable) for accessing the list on different devices.
- Offline access to the Notebook with sync-up when online.

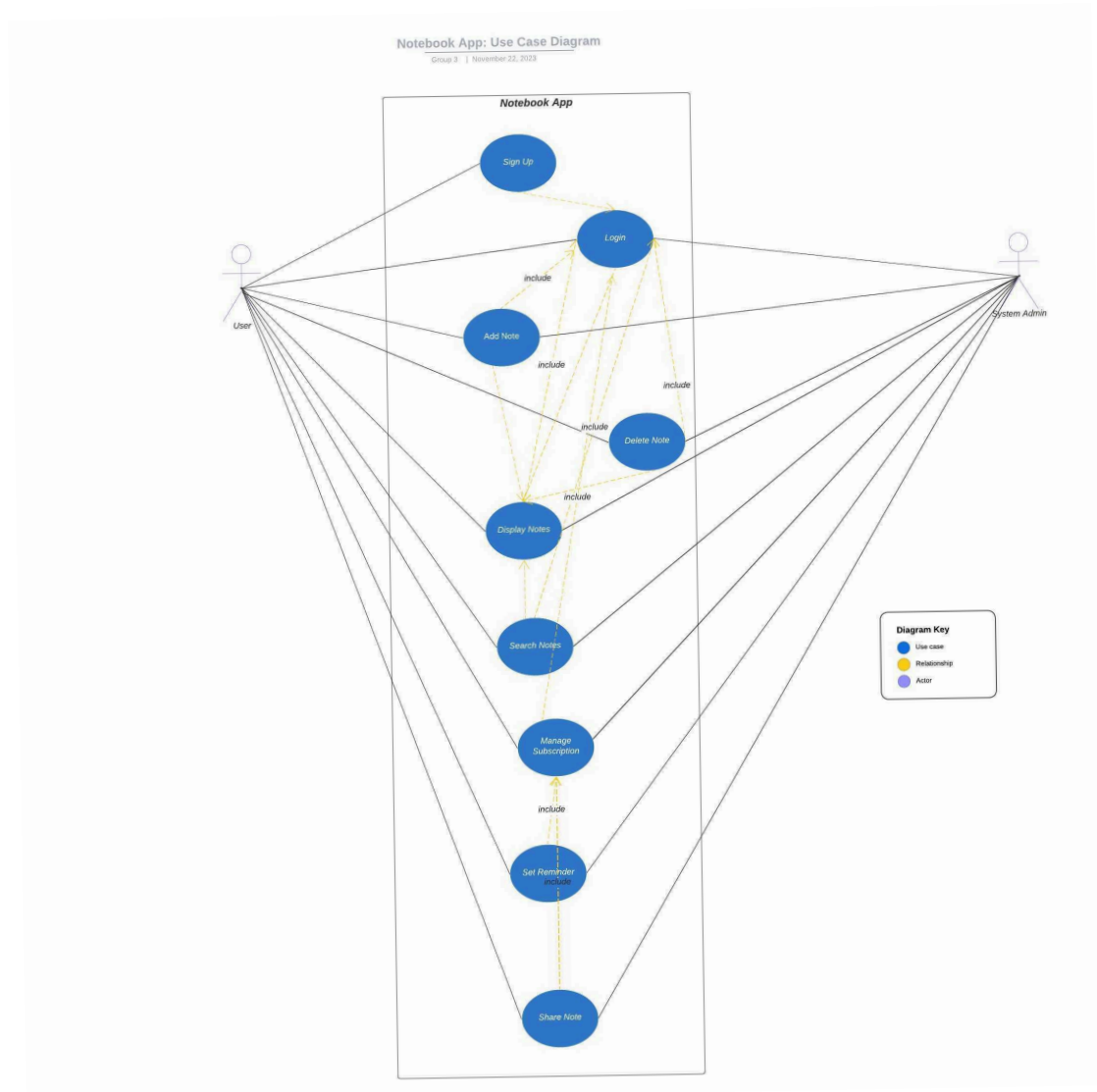


Fig.01. Use-case Diagram

4.1 System Design

4.1.1 Context Diagram:

In software engineering, a context diagram is a simple graphical representation that shows the system under development in the context of its environment and interactions. For a Notebook project, the context diagram plays a crucial role in visually depicting how the system interfaces with users and other systems.

Components of the Context Diagram:

Notebook System (Central Node):

- Represented as the central circle or square in the diagram.
- This is the core software system that you're developing.

Users (External Entities):

- Illustrated as rectangles or ovals around the central system.
- Can include different types of users like individual users, team leaders, or administrators.

External Systems (External Entities):

- Other systems that interact with the Notebook system.
- Examples might include email services, or notification systems.

Data Flow (Arrows):

- Arrows show the flow of information between the system and external entities.
- They indicate how users interact with the system (e.g., adding tasks, receiving notifications) and how the system communicates with other software.

Scenario in the Context Diagram:

- A user (external entity) interacts with the Notebook system by adding a new task. This is shown as an arrow from the user to the system.
- The system then integrates with a calendar application (another external entity) to set reminders for the task, depicted by an arrow from the system to the calendar application.

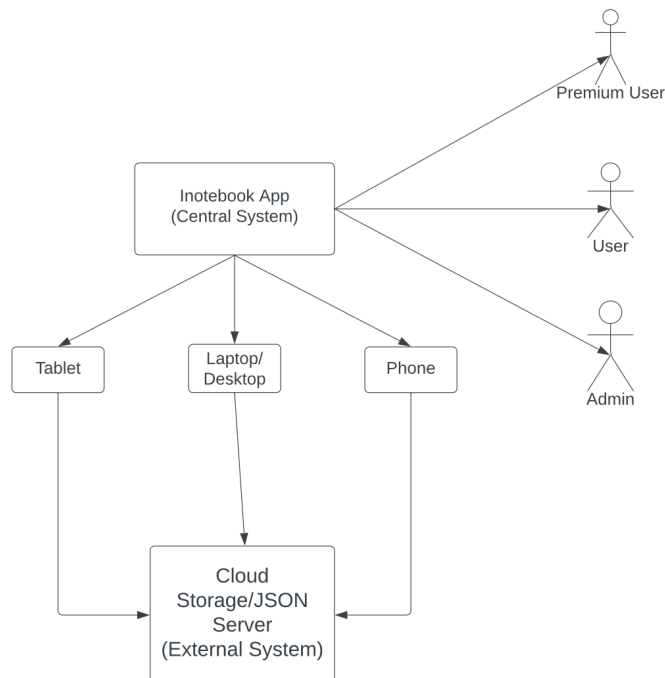


Fig.02.Context Diagram

4.1.2 Class Diagram:

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. For a Notebook project, a class diagram will play a crucial role in outlining the object-oriented structure of the system. Here's a brief overview of what a class diagram for a Notebook project might include:

Classes:

- Each class represents a different entity or concept in the Notebook application.
- Example classes could be User, Task, List, Reminder, etc.

Attributes:

- Attributes are the properties or characteristics of each class.
- For instance, the User class might have attributes like username, email, and password; the Task class could have title, description, dueDate, etc.

Operations/Methods:

- These are the functions or behaviors that each class can perform.
- The User class might have methods like login(), logout(), createList(), etc., while the Task class might include methods like markAsComplete(), editTask(), etc.

Relationships:

- The diagram will show how the classes are related to each other.
- Common relationships include associations (simple links between two classes), aggregations (representing whole-part relationships), and inheritances (where one class is a subclass of another).

Multiplicity:

- This specifies how many instances of a class can be associated with one instance of another class.
- For example, a User might have "1 too many" relationships with List, indicating a user can have multiple lists.

Role of the Class Diagram in the Notebook Project:

Structural Overview:

- Provides a clear overview of the system's structure and the relationships between different parts of the system.
- Essential for understanding how the system is organized before delving into the code.

Guidance for Development:

- Serves as a blueprint for developers, showing what needs to be implemented in the code.
- Helps in identifying required classes, their attributes, and methods.

Facilitating Maintenance and Scalability:

Enhancing Communication Among Team Members:

- Useful tool for communication within the development team and with other stakeholders (like system architects and designers).

Elements in the Notebook Class Diagram:

- A Task class with attributes like taskID, taskName, status, and methods like updateStatus(), setReminder().
- A User class linked to a List class, indicating that each user can have one or more lists.
- An authentication class for managing user login and registration processes.

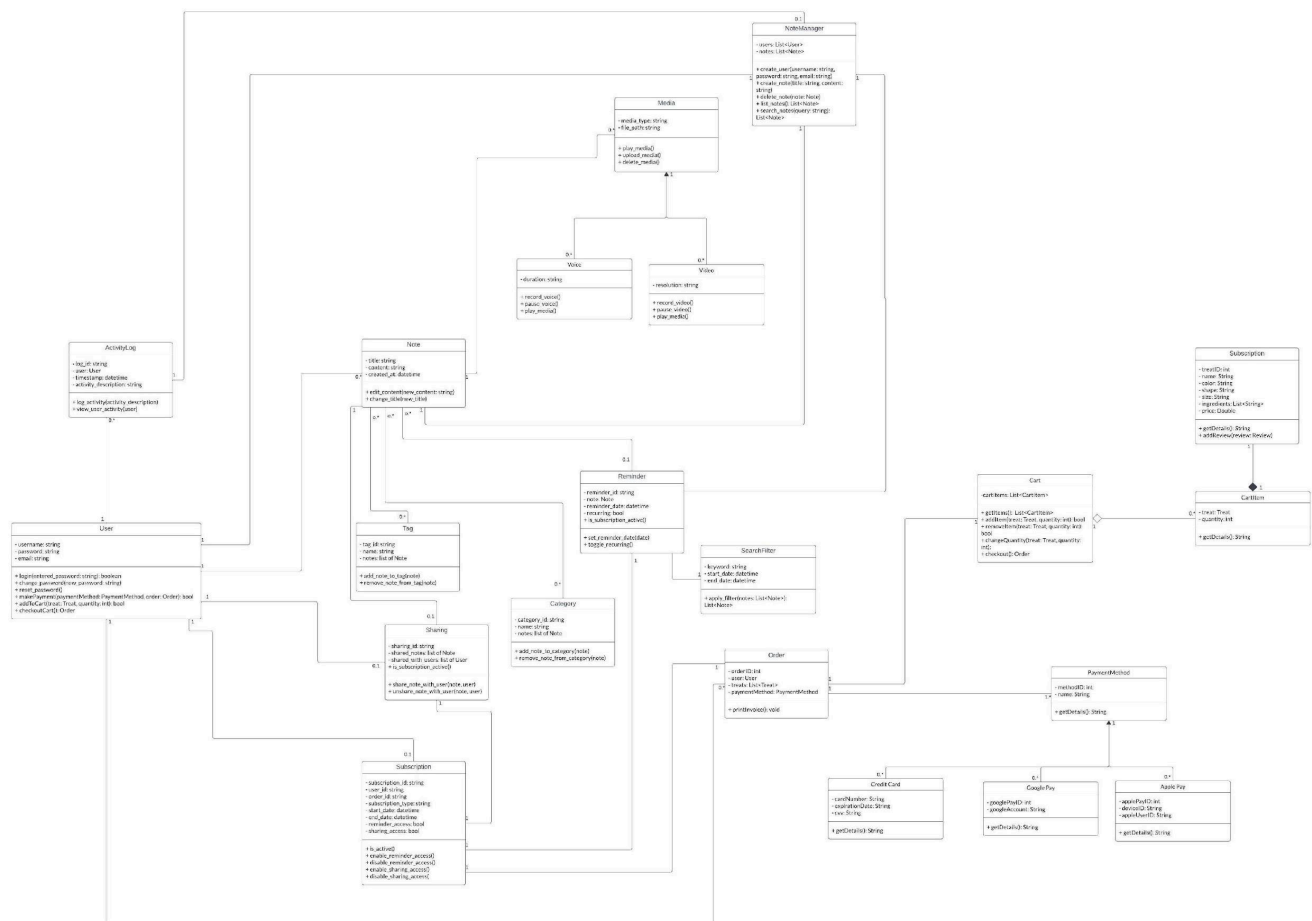


Fig.03.Class Diagram

4.1.3 System Architecture Diagram:

The system architecture diagram plays a crucial role in depicting the overall structure of the system, including its major components and the interactions between them. This diagram is essential for understanding how the system is organized at a high level. Here's a brief overview of what a system architecture diagram for a Notebook project might include:

Components of the System Architecture Diagram:

User Interface (UI):

- This layer represents the front-end of the application, through which users interact with the system.
- It includes screens for task management, user account settings, notifications, etc.

Application Layer:

- This layer contains the business logic of the system.
- It processes user requests, manages tasks and user data, handles notifications, etc.

Database Layer:

- This layer is responsible for data storage and retrieval.
- It stores information about users, tasks, lists, reminders, etc.

Server/Backend Layer:

- The server hosts the application and handles requests from the client (UI).
- It might include a web server, application server, and the necessary APIs for communication between the front-end and back-end.

Integration Points:

- Points where the Notebook system integrates with external systems, such as email servers for notifications, calendar applications, or cloud storage services.

Security Layer:

- This includes security measures to protect data and user privacy, like authentication mechanisms, encryption, and secure data transmission protocols.

Role of the System Architecture Diagram:

High-Level Overview:

- Provides a bird's eye view of the entire system, illustrating how different components fit together.
- Essential for initial planning and designing the system.

Guidance for Development and Deployment:

- Serves as a guide for developers and system engineers in building and deploying the system.

- Helps in understanding the technical requirements and infrastructure needs.

Facilitating Communication:

- Useful in communicating the structure and design of the system to stakeholders, including developers, project managers, and clients.

Identifying Potential Issues:

- Helps in identifying potential bottlenecks, dependencies, and challenges in the system architecture.

Scenario in the System Architecture Diagram:

- The diagram might show a web application with a front-end built in a framework like React, communicating through APIs with a back-end server running Node.js.
- The server interacts with a database such as PostgreSQL to store and retrieve task data.
- Integration points with external services like Google Calendar or a notification service are also depicted.

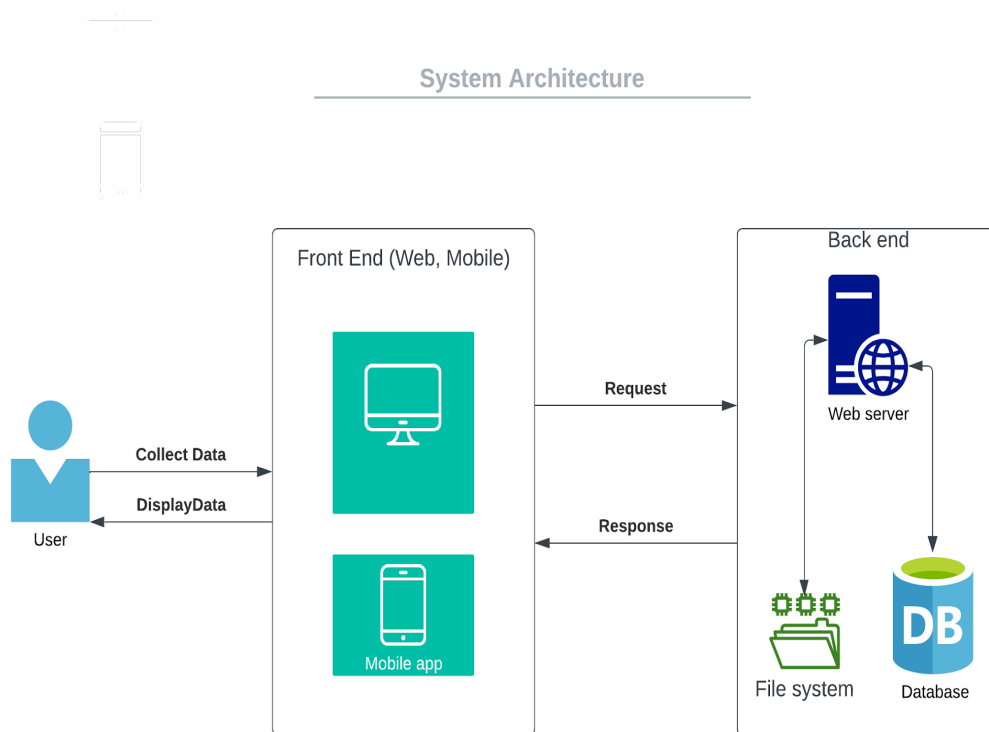


Fig.04. System Architecture Diagram

4.1.4. Activity Diagram

User Registration/Login: The first activity is for users to either register for a new account or log in to an existing one.

View Tasks: Once logged in, users can view their list of tasks. This might involve filtering or sorting tasks based on criteria like due date, priority, or category.

Add Task: Users can add new tasks to their list. This involves entering details such as the task name, description, due date, and priority.

Edit Task: Users have the option to edit the details of existing tasks. This can include changing the task name, description, due date, priority, or marking the task as completed.

Delete Task: Users can delete tasks from their list.

Mark Task as Completed: Users can mark tasks as completed. This might move the task to a completed list or simply strike it out on the current list.

Notifications: The system may send reminders or notifications to the user about upcoming deadlines or pending tasks.

Logout: Finally, users can log out of the application.

Each of these activities represents a step in the workflow of the "Notebook" software, and they are typically interconnected.

4.1.5 Data Flow Diagram

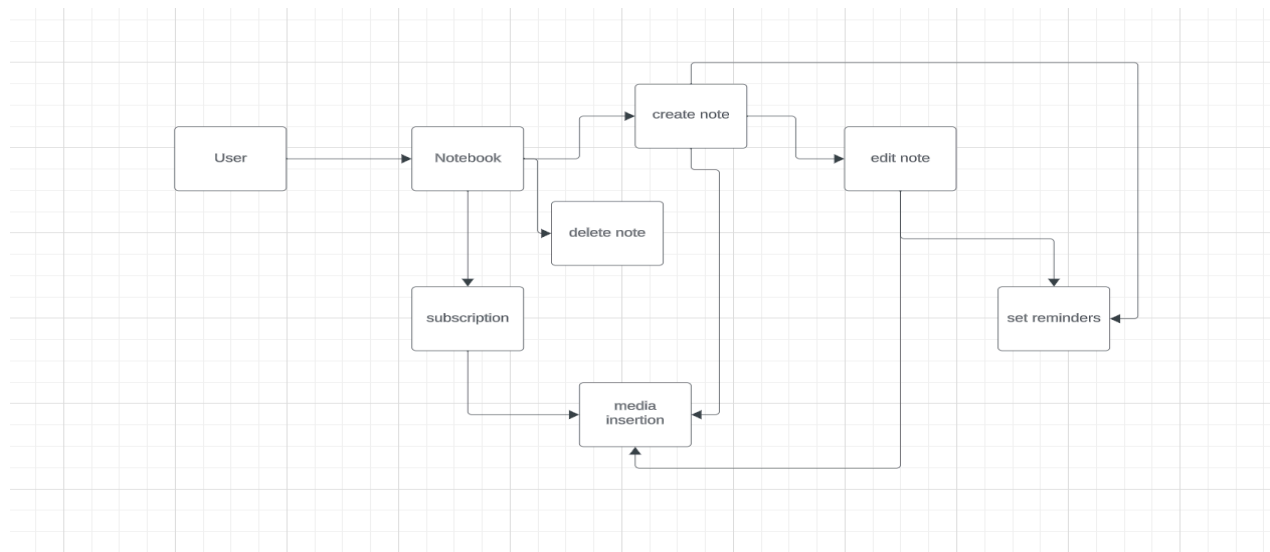


Fig.06. Data Flow Diagram

5. Specific Requirements:

5.1 Hardware requirements:

- Intel Core Processor
- Ram 4GB and above
- 64- or 32-bit windows

5.2 Software Requirements:

- Environment: React.js
- Languages: React, HTML & CSS
- Database: Json structure

6. Implementation:

6.1 Project Overview:

The Notebook application, designed as a part of our software engineering project, aims to provide a seamless and efficient task management experience to users. The application has been developed with a focus on user-friendliness, reliability, and scalability.

6.2 Technology Stack:

- Frontend: We utilized React.js for the front-end to create a responsive and dynamic user interface. This choice was driven by React's component-based architecture, which allows for reusable UI components.
- Backend: Node.js with Express.js framework was employed for the backend. This combination was chosen for its scalability and its efficient handling of asynchronous requests.
- Database: MongoDB database was used for its flexibility and performance in handling large volumes of unstructured data.
- Authentication: JWT (JSON Web Tokens) was implemented for secure user authentication.
- Deployment: The application was containerized using Docker and deployed on AWS (Amazon Web Services) for high availability and scalability.

6.3 Key Features:

- Task Management: Users can create, edit, and delete tasks, set priorities, and deadlines.
- List Organization: Enables users to categorize tasks into custom lists.
- Reminders and Notifications: Integrated functionality to set reminders and receive notifications.
- Search and Filter: Allows users to search for tasks and filter them based on various criteria like due date, priority, etc.
- User Account Management: Features user registration, login, and profile management.

6.4 Development Approach:

- Agile Methodology: The project was managed using Agile principles, with regular sprints and iterative development. This approach facilitated flexibility in development and incorporating user feedback.
- Version Control: Git was used for version control, with GitHub as the repository hosting service, ensuring effective team collaboration and code management.
- Testing: The application underwent various testing phases, including unit testing, integration testing, and user acceptance testing (UAT), to ensure functionality and reliability.

6.5 Challenges Encountered:

- Cross-Platform Compatibility: Ensuring consistent user experience across various devices and browsers was challenging.
- Performance Optimization: Balancing between feature-richness and application performance required meticulous code optimization and efficient database querying.
- Security Concerns: Implementing robust security measures to protect user data and prevent vulnerabilities.

6.6 CODE

6.6.1 App.js

```
import './App.css';
import React from 'react'
import {
  BrowserRouter as Router,
  Routes,
  Route
} from "react-router-dom";
import Notes from './pages/Notes';
import Create from './pages/Create';
import { createTheme, ThemeProvider } from '@mui/material'
import { red } from '@mui/material/colors';
import Layout from './components/Layout';
```

```
function App() {

  const theme = createTheme({
    palette: {
      secondary: red
    }
  })
  return (
    <ThemeProvider theme={theme}>
      <Router >
        <Layout>
          <Routes>
            <Route path="/" element={<Notes />}>
            </Route>
            <Route path="/create" element={<Create />}>
            </Route>
          </Routes>
        </Layout>
      </Router >
    </ThemeProvider>
  );
}
```

```
export default App;
```

6.6.2 App.css

```
.App {
  text-align: center;
```

```

}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

6.6.3 index.js

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />

```

```

    </React.StrictMode>
  );

```

```

reportWebVitals();

```

6.6.4 index.css

```

body {
  margin: 0;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

```

6.6.5 component/layout.js

```

import { Avatar, Box, Typography } from '@mui/material'
import React from 'react'
import SideNav from './SideNav'
import AppBar from '@mui/material/AppBar';
import Toolbar from '@mui/material/Toolbar';
import { format } from 'date-fns';

```

```

export default function Layout(props) {

```

```

  return (
    <Box sx={{ display: 'flex' }}>
      <AppBar sx={{ width: `calc(100% - 240px)` }} color="secondary" elevation={0}>
        <Toolbar >
          <Typography sx={{ flexGrow: 1 }}>
            Today is the {format(new Date(), 'do MMMM Y')}
          </Typography>
          <Typography>
            Dev    {/* Login and Logout Button */}
          </Typography>
          <Avatar src='/ava.jpg' sx={{
            marginLeft: {
              sm: 2
            }
          }} />
        </Toolbar>
      </AppBar>
      <Box>
        <SideNav />
      </Box>
      <Box sx={{
        backgroundColor: '#f9f9f9',
        width: '100%',
        padding: {
          sm: 3
        }
      }} />
    </Box>
  )
}

```

```

    }
  }}>
    <Box sx={theme => theme.mixins.toolbar} />
    {props.children}
  </Box>
</Box >

)
}

```

6.6.6 component/NoteCard.js

```

import React from 'react'
import { Card, CardHeader, CardContent, Typography, Avatar } from '@mui/material';
import { IconButton } from '@mui/material';
import { DeleteOutlined } from '@mui/icons-material';
import { blue, green, pink, yellow } from '@mui/material/colors';

export default function NoteCard({ note, handleDelete }) {
  return (
    <Card >
      <CardHeader
        avatar={
          <Avatar sx={{
            backgroundColor: blue[500],
            ...(note.category === 'work' && {
              backgroundColor: yellow[700],
            }),
            ...(note.category === 'money' && {
              backgroundColor: green[500],
            }),
            ...(note.category === 'todos' && {
              backgroundColor: pink[500],
            })
          }} > {note.category[0].toUpperCase()}</Avatar>
        }
        action={
          <IconButton onClick={() => handleDelete(note.id)}>
            <DeleteOutlined />
          </IconButton >
        }
        title={note.title}
        subheader={note.category.charAt(0).toUpperCase() + note.category.slice(1)}
      />
      <CardContent>
        <Typography variant='body2'>
          {note.details}
        </Typography>

```

```

        </CardContent>
      </Card >
    )
  }

```

6.6.7 SideNav.js

```

import { Drawer, List, ListItem, ListItemIcon, ListItemText, ListItemButton, Typography } from
 '@mui/material'
import AddCircleOutlineOutlinedIcon from '@mui/icons-material/AddCircleOutlineOutlined';
import SubjectOutlinedIcon from '@mui/icons-material/SubjectOutlined';
import React from 'react'
import { useNavigate, useLocation } from 'react-router-dom';

```

```

const drawWidth = 240

```

```

export default function SideNav() {

```

```

  const navItems = [
    {
      text: "My Notes",
      icon: <SubjectOutlinedIcon color='secondary' />,
      path: '/'
    },
    {
      text: "Create Note",
      icon: <AddCircleOutlineOutlinedIcon color='secondary' />,
      path: '/create'
    }
  ]

```

```

  const navigate = useNavigate()
  const location = useLocation()

```

```

  const checkActive = (path) => {
    if (path === location.pathname) {
      return "#f4f4f4"
    }
  }

```

```

  return (

```



```

<div>
  <Drawer sx={{
    width: drawWidth,
  }}
  variant={'permanent'}
  anchor={'left'}
  PaperProps={{
    sx: {
      width: drawWidth
    }
  }}
  >
    <div>

      <Typography variant='h5' sx={{
        padding: {
          sm: 2
        }
      }}>
        Soft Eng Group3 Notes
      </Typography>
    </div>
    <div>
      <List>
        {navItems.map(item => (
          <ListItemButton key={item.text} sx={{ backgroundColor:
checkActive(item.path) }}>
            <ListItem onClick={() => navigate(item.path)}>
              <ListItemIcon>{item.icon}</ListItemIcon>{console.log(item.path)}
              <ListItemText primary={item.text} />
            </ListItem>
          </ListItemButton>
        ))}
      </List>
    </div>

  </ Drawer >
</div>
)
}

```

6.6.8. pages/Create.js

```

import React, { useState } from 'react'
import { Container, Typography, Button, TextField, FormControl, FormLabel, FormControlLabel,
Radio, RadioGroup } from '@mui/material'

```

```

import ArrowForwardIcon from '@mui/icons-material/ArrowForward';
import { useNavigate } from 'react-router-dom';

export default function Create() {

  const navi = useNavigate()
  const [title, setTitle] = useState("");
  const [details, setDetails] = useState("");
  const [titleErr, setTitleErr] = useState(false);
  const [detailsErr, setDetailsErr] = useState(false);
  const [category, setCategory] = useState('money')


  const handleSubmit = (e) => {
    e.preventDefault()
    if (title) {
      setTitleErr(false)
    } else {
      setTitleErr(true)
    }
    if (details) {
      setDetailsErr(false)
    } else {
      setDetailsErr(true)
    }

    if (title && details) {
      fetch('http://localhost:8000/notes', {
        method: 'POST',
        headers: { "Content-type": "application/json" },
        body: JSON.stringify({ title, details, category })
      }).then(() => navi('/'))
    } else {
      console.log('false')
    }
  }

  return (
    <Container>
      <Typography
        variant='h6'
        color='textSecondary'
        gutterBottom
      >
        Create a new Note
      </Typography>

      <form noValidate autoComplete='off' onSubmit={handleSubmit}>
        <TextField

```

```

      onChange={(e) => setTitle(e.target.value)}
      sx={{
        my: 3,
        display: 'block'
      }}
      label='Note Title'
      variant='outlined'
      color='secondary'
      fullWidth
      required
      error={titleErr}
    />
    <TextField
      onChange={(e) => setDetails(e.target.value)}
      sx={{
        my: 3,
        display: 'block'
      }}
      label='Details'
      variant='outlined'
      color='secondary'
      fullWidth
      required
      multiline
      rows={5}
      error={detailsErr}
    />

    <FormControl
      sx={{
        my: 3,
        display: 'block'
      }}
    >
      <FormLabel>Note Category</FormLabel>
      <RadioGroup value={category} onChange={(e) => { setCategory(e.target.value) }}>
        <FormControlLabel control={<Radio color='secondary' />} value='money'
label='Money' />
        <FormControlLabel control={<Radio color='secondary' />} value='todo' label='To
Do' />
        <FormControlLabel control={<Radio color='secondary' />} value='reminders'
label='Reminders' />
        <FormControlLabel control={<Radio color='secondary' />} value='work'
label='Work' />
      </RadioGroup>
    </FormControl>

    <Button

```

```

        type='submit'
        color='secondary'
        variant='contained'
        endIcon={<ArrowForwardIcon />}
      >
        Submit
      </Button>
    </form>
  </Container>
)
}

```

6.6.9. pages/Notes.js

```

import { Container, Grid } from '@mui/material'
import React, { useEffect, useState } from 'react'
import NoteCard from '../components/NoteCard'
import { Masonry } from '@mui/lab'

```

```

export default function Create() {

```

```

  const [notes, setNotes] = useState([])
  useEffect(() => {
    fetch('http://localhost:8000/notes')
      .then(res => res.json())
      .then(data => setNotes(data))
  }, [])
  const handleDelete = async (id) => {
    await fetch('http://localhost:8000/notes/' + id, {
      method: 'DELETE'
    })
    const newNotes = notes.filter(note => note.id !== id)
    setNotes(newNotes)
  }

```

```

  return (
    <Container>
      <Masonry columns={3} spacing={2}>
        {notes.map(note => (
          <Grid item key={note.id} md={6} lg={4} xs={12}>
            <NoteCard note={note} handleDelete={handleDelete} />
          </Grid>
        ))}
      </Masonry>
    </Container>
  )

```

```

    )
  }

```

6.6.10. RegisterPage.js

```

import React from 'react'
import { Link } from 'react-router-dom'

```

```

import '../App.css'

```

```

export default function SignUpPage() {

```

```

  return (
    <div className="text-center m-5-auto">
      <h2>Join us</h2>
      <h5>Create your personal account</h5>
      <form action="/home">
        <p>
          <label>Username</label><br/>
          <input type="text" name="first_name" required />
        </p>
        <p>
          <label>Email address</label><br/>
          <input type="email" name="email" required />
        </p>
        <p>
          <label>Password</label><br/>
          <input type="password" name="password" required />
        </p>
        <p>
          <input type="checkbox" name="checkbox" id="checkbox" required /> <span>I
agree all statements in <a href="https://google.com" target="_blank" rel="noopener
noreferrer">terms of service</a></span>.
        </p>
        <p>
          <button id="sub_btn" type="submit">Register</button>
        </p>
      </form>
      <footer>
        <p><Link to="/">Back to Homepage</Link>.</p>
      </footer>
    </div>
  )

```

```

}

```

6.6.11. LoginPage.js

```
import React from 'react'
import { Link } from 'react-router-dom'

import '../App.css'

export default function SignInPage() {
  return (
    <div className="text-center m-5-auto">
      <h2>Sign in to us</h2>
      <form action="/home">
        <p>
          <label>Username or email address</label><br/>
          <input type="text" name="first_name" required />
        </p>
        <p>
          <label>Password</label>
          <Link to="/forget-password"><label className="right-label">Forget
password?</label></Link>
          <br/>
          <input type="password" name="password" required />
        </p>
        <p>
          <button id="sub_btn" type="submit">Login</button>
        </p>
      </form>
      <footer>
        <p>First time? <Link to="/register">Create an account</Link>.</p>
        <p><Link to="/">Back to Homepage</Link>.</p>
      </footer>
    </div>
  )
}
```

6.6.12 LandingPage.js

```
import React from 'react'
import { Link } from 'react-router-dom'

import '../App.css'
import BackgroundImage from '../assets/images/bg.png'
```

```

export default function LandingPage() {
  return (
    <header style={ HeaderStyle }>
      <h1 className="main-title text-center">login / register page</h1>
      <p className="main-para text-center">join us now and don't waste time</p>
      <div className="buttons text-center">
        <Link to="/login">
          <button className="primary-button">log in</button>
        </Link>
        <Link to="/register">
          <button className="primary-button" id="reg_btn"><span>register
</span></button>
        </Link>
      </div>
    </header>
  )
}

```

```

const HeaderStyle = {
  width: "100%",
  height: "100vh",
  background: `url(${BackgroundImage})`,
  backgroundPosition: "center",
  backgroundRepeat: "no-repeat",
  backgroundSize: "cover"
}

```

6.6.13 reportWebVitals.js

```

const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

```

```
export default reportWebVitals;
```

6.6.14. setupTests.js

```
import '@testing-library/jest-dom';
```

6.7 Results Screenshots:

6.7.1 setup localhost for the application

```
josejoe025@joe-workstation:~/Downloads/notes-master$ npm install -g json-server
added 116 packages in 2s
15 packages are looking for funding
  run `npm fund` for details
josejoe025@joe-workstation:~/Downloads/notes-master$ json-server --watch data/db.json --port 8000

\{^_^}/ hi!

Loading data/db.json
Done

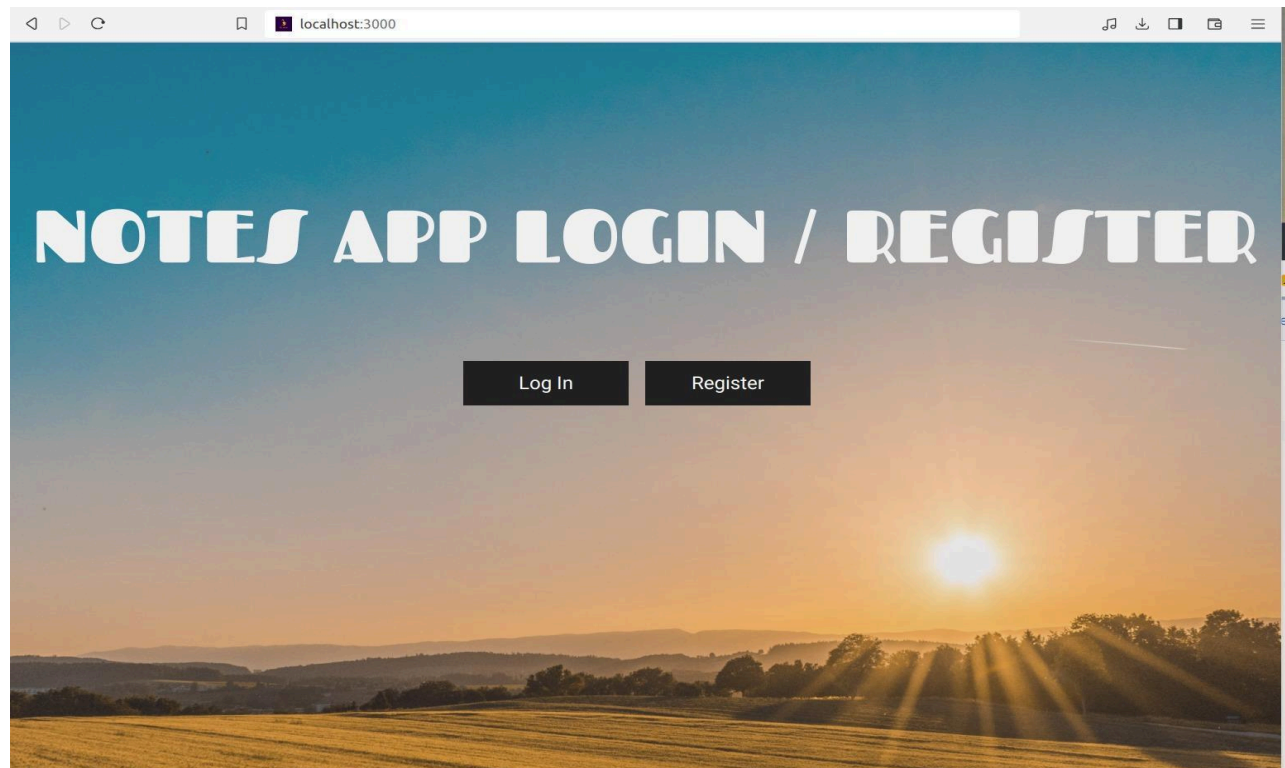
Resources
http://localhost:8000/notes

Home
http://localhost:8000

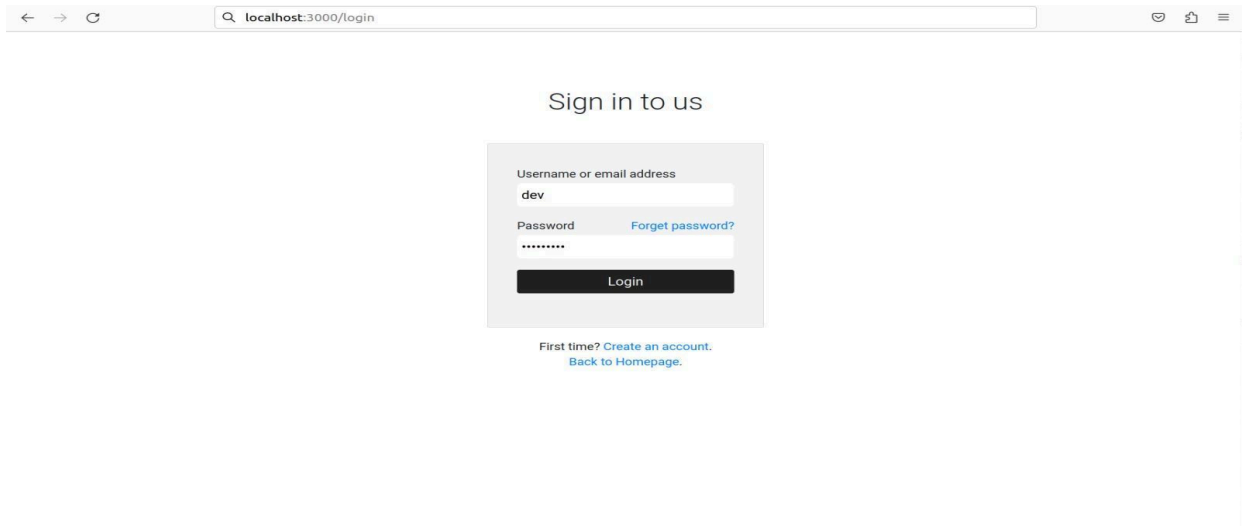
Type s + enter at any time to create a snapshot of the database
Watching...

GET /notes 200 5.855 ms - -
GET /notes 304 3.291 ms - -
GET /notes 200 2.424 ms - -
GET /notes 200 3.131 ms - -
POST /notes 201 18.953 ms - 81
GET /notes 200 2.908 ms - -
GET /notes 304 2.722 ms - -
DELETE /notes/1 200 4.460 ms - 2
DELETE /notes/2 200 3.462 ms - 2
GET /notes 200 1.842 ms - 932
GET /notes 304 3.535 ms - -
DELETE /notes/5 200 3.455 ms - 2
POST /notes 201 4.165 ms - 81
GET /notes 200 2.920 ms - 907
GET /notes 304 2.713 ms - -
DELETE /notes/8 200 3.184 ms - 2
```

6.7.2. Landing page for the application:

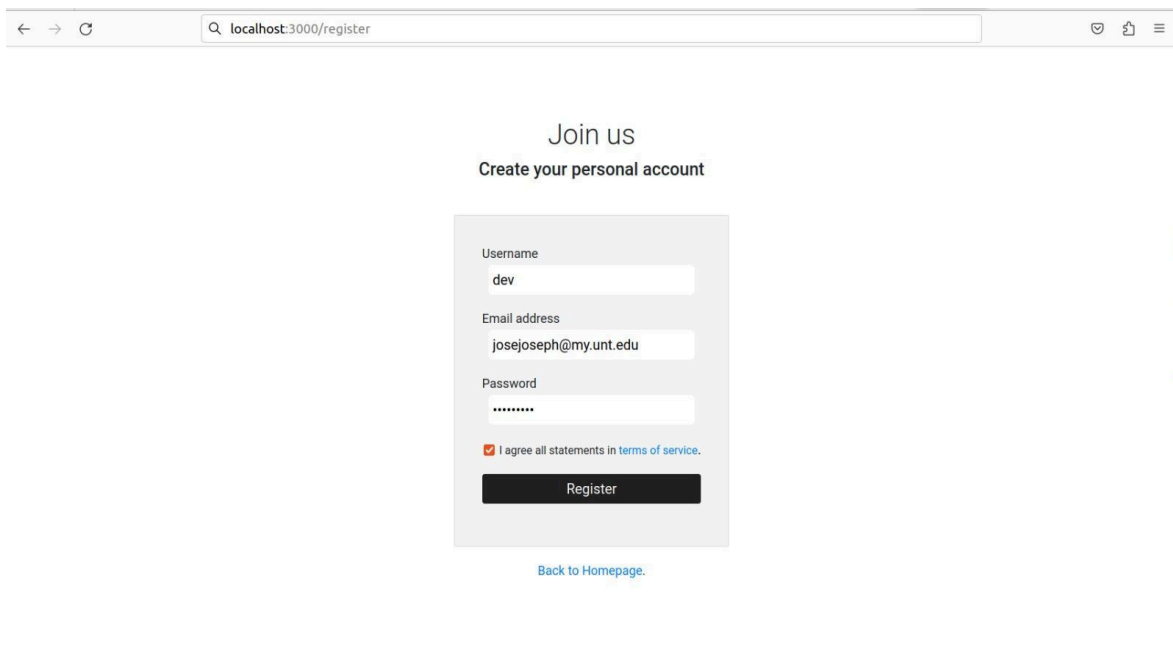


6.7.3. Login Page



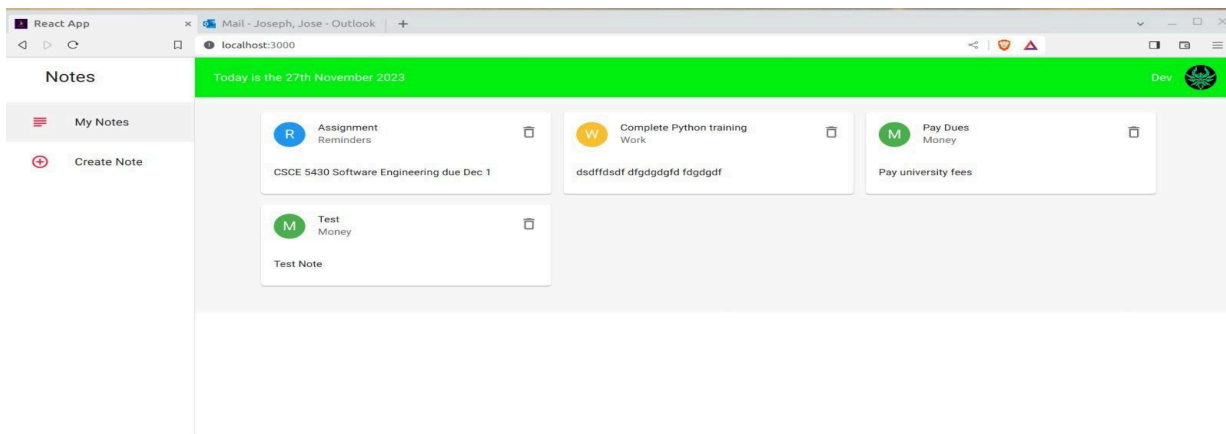
A screenshot of a web browser showing the login page at `localhost:3000/login`. The page has a clean, minimalist design with a light gray background. At the top, there's a navigation bar with a search icon, a back arrow, a forward arrow, and a refresh icon. The main heading is "Sign in to us" in a dark gray font. Below this, there's a white rectangular form box. Inside the box, the first field is labeled "Username or email address" and contains the text "dev". The second field is labeled "Password" and contains a series of dots. To the right of the password field is a blue link that says "Forgot password?". At the bottom of the form box is a black button with the word "Login" in white. Below the form box, there's a line of text that says "First time? [Create an account.](#)" and a blue link that says "[Back to Homepage.](#)".

6.7.4. Register page



A screenshot of a web browser showing the register page at `localhost:3000/register`. The page has a clean, minimalist design with a light gray background. At the top, there's a navigation bar with a search icon, a back arrow, a forward arrow, and a refresh icon. The main heading is "Join us" in a dark gray font, followed by the subheading "Create your personal account" in a smaller, bold, dark gray font. Below this, there's a white rectangular form box. Inside the box, there are three fields: "Username" with the text "dev", "Email address" with the text "josejoseph@my.unt.edu", and "Password" with a series of dots. Below the password field is a checkbox that is checked, followed by the text "I agree all statements in [terms of service.](#)". At the bottom of the form box is a black button with the word "Register" in white. Below the form box, there's a blue link that says "[Back to Homepage.](#)".

6.7.5. Dashboard of Notebook



6.7.6. Data of dashboard in Json file



6.7.7 Create Note

The screenshot shows a web browser window with the URL `localhost:3000/create`. The application has a green header bar with the text "Today is the 27th November 2023" and a "Dev" button. On the left, there is a sidebar with "Notes" and two buttons: "My Notes" and "Create Note". The main area is titled "Create a new Note" and contains a form with the following fields:

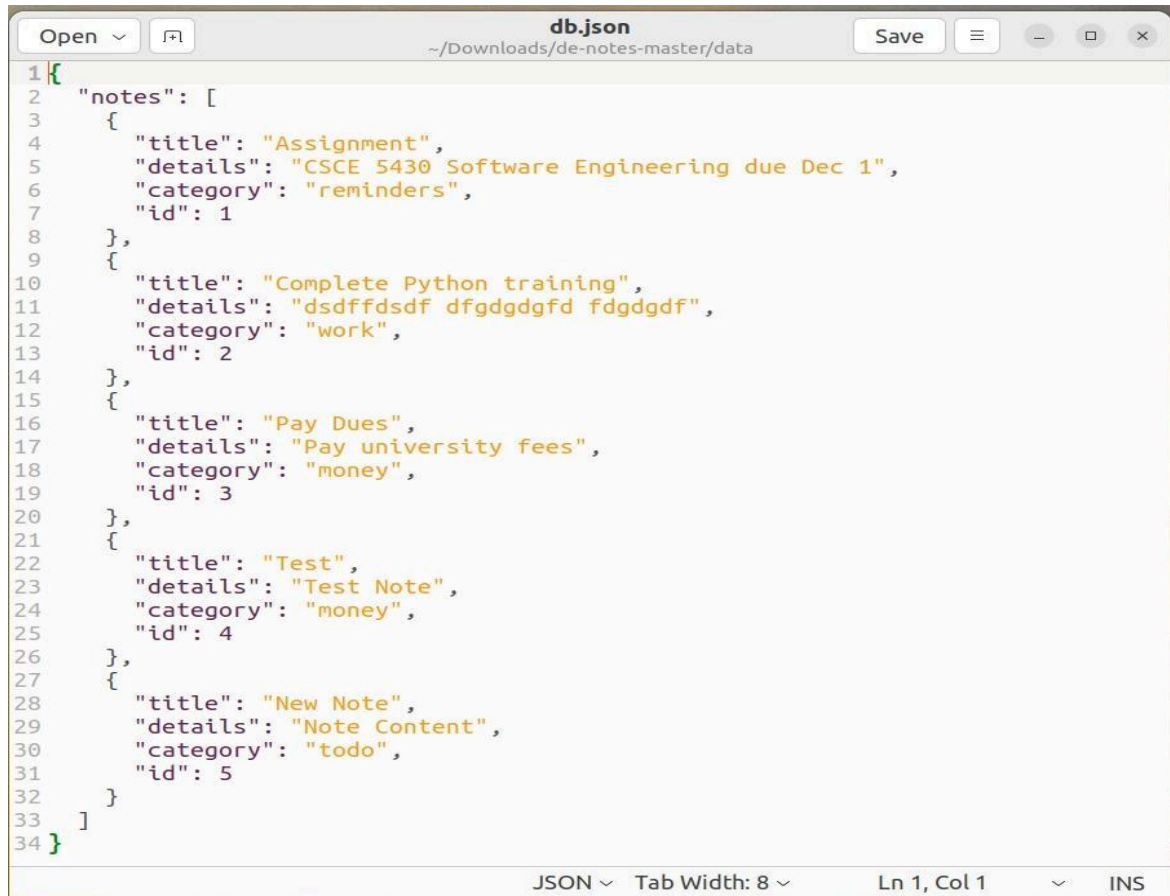
- Note Title ***: A text input field with the placeholder text "New Note".
- Details ***: A large text area with the placeholder text "Note Content".
- Note Category**: A group of radio buttons with the following options:
 - ☒ Money
 - ☐ To Do
 - ☐ Reminders
 - ☐ Work
- SUBMIT**: A green button with a right-pointing arrow.

6.7.8 View and display the new create note in dashboard

The screenshot shows the same web application at the `localhost:3000` URL. The sidebar now shows "My Notes" as the active button. The main area displays a list of notes in a grid format. Each note card has a colored circular icon, a title, a category, and a content area. The notes are:

- Complete Python training**: Category "Work", content "dsdffdsdf dfgdgdgfd fgdgdgfd".
- Pay Dues**: Category "Money", content "Pay university fees".
- Test**: Category "Money", content "Test Note".
- New Note**: Category "To Do", content "Note Content".

6.7.9. Newly create note data in Json file format



```
1 {
2   "notes": [
3     {
4       "title": "Assignment",
5       "details": "CSCE 5430 Software Engineering due Dec 1",
6       "category": "reminders",
7       "id": 1
8     },
9     {
10      "title": "Complete Python training",
11      "details": "dsdffd sdf dfgdgdgfd fdgdgdf",
12      "category": "work",
13      "id": 2
14    },
15    {
16      "title": "Pay Dues",
17      "details": "Pay university fees",
18      "category": "money",
19      "id": 3
20    },
21    {
22      "title": "Test",
23      "details": "Test Note",
24      "category": "money",
25      "id": 4
26    },
27    {
28      "title": "New Note",
29      "details": "Note Content",
30      "category": "todo",
31      "id": 5
32    }
33  ]
34 }
```

This way users can Create new Note and we can see that entered data in the Json File. In the similar way we can Delete the Notes and edit them and display them in the dashboard. To access premium features, users can subscribe to and proceed to the payment gateway and pay for the application subscription for monthly or annual plans.

7. Testing

Creating comprehensive test cases for a Notebook project in software engineering involves covering various aspects of the application to ensure its functionality, reliability, usability, and security.

Functional Test Cases:

A functionality test case for an application is a specific scenario designed to check whether a particular function or feature of the application works as intended. It usually involves the following elements:

1. User Registration:

- Test successful user registration with valid credentials.
- Test registration with an already existing email/username.

- Test registration with invalid email formats and insufficient password strength.

2. User Login:

- Test login with valid credentials.
- Test login with invalid credentials (incorrect password or username).
- Test the response when the user attempts to log in with an unregistered account.

3. Password Recovery:

- Test the password recovery process with a registered email.
- Test the process with an unregistered email.

4. Adding a New Task:

- Test adding a task with all required fields filled.
- Test adding a task with missing mandatory fields.
- Test the limit for the number of characters in the task description.

5. Editing a Task:

- Test editing the details of an existing task.
- Test saving changes and ensuring they persist.

6. Deleting a Task:

- Test the deletion of a task.
- Test if the task is removed from the list and database.

7. Setting Task Priorities and Deadlines:

- Test setting and changing task priorities (high, medium, low).
- Test setting deadlines and reminders for tasks.

8. Task Completion:

- Test marking a task as complete and incomplete.
- Test visibility and sorting of completed vs. incomplete tasks.

9. List Management:

- Test creating, editing, and deleting task lists.
- Test organizing tasks into different lists.

10. Search Functionality:

- Test the search feature for tasks and lists.
- Test search accuracy and response time.

Usability Test Cases:

A usability test case in the context of software development and user experience design is a specific scenario or task that is designed to evaluate the usability of a system, application,

website, or product. This test case focuses on how easily and effectively a user can interact with the interface and complete desired tasks. The key components of a usability test case typically include:

1. Navigation:

- Test the ease of navigation between different sections of the application.
- Test the intuitiveness of the user interface.

2. Responsiveness:

- Test the application's responsiveness on different devices and screen sizes.

3. Accessibility:

- Test for accessibility features like font size adjustments, color contrast, and keyboard navigation.

Performance Test Cases:

A performance test case in the context of software development and quality assurance is a specific scenario or set of tasks designed to evaluate and measure the performance of a software application or system under a particular workload. Performance testing is aimed at ensuring that the application responds quickly, can handle the expected number of users, and remains stable under stress. Key components of a performance test case include:

1. Load Testing:

- Test the application's performance under heavy load (many simultaneous users).

2. Stress Testing:

- Test how the application behaves under extreme conditions (e.g., a very large number of tasks).

3. Speed Testing:

- Test the response time for various actions (e.g., adding or deleting tasks).

Security Test Cases:

A security test case in software development and testing is a specific set of procedures designed to evaluate and ensure the security aspects of a software application or system. The goal of security testing is to identify vulnerabilities, threats, and risks in the software and to ensure that data and resources are protected from potential breaches. Key components of a security test case typically include:

1. Data Encryption:

- Test if sensitive data (like passwords) is properly encrypted.

2. SQL Injection:

- Test the application's vulnerability to SQL injection attacks.

3. Cross-Site Scripting (XSS):

- Test for XSS vulnerabilities in input fields.

4. Authentication:

- Test the security of the authentication process (e.g., against brute force attacks).

5. Authorization:

- Test if users can access or modify tasks/lists that they shouldn't have access to.

Integration Test Cases:

An integration test case in software development is a specific type of test designed to evaluate how different modules or components of a software application interact with each other. The primary goal of integration testing is to identify issues related to the interaction between integrated units or systems, ensuring that they work together seamlessly as intended. Key components of an integration test case typically include:

1. Third-Party Integrations:

- Test the integration with external services (like calendar apps, email services).

2. API Connectivity:

- Test the reliability and response time of API connections.

These test cases cover a wide range of aspects including functionality, usability, performance, security, and integration. They are crucial for ensuring that the Notebook application is robust, reliable, and user-friendly. This comprehensive approach to testing will help in identifying and resolving potential issues before the application is released to end users.

8. Deployment:

Creating a deployment diagram for a Notebook project in a textual format can provide a clear layout of how the components are deployed in the system's architecture. However, it's important to note that an actual diagram would typically be created using a visual tool like UML modeling software.

Deployment Diagram Description

Nodes:

- **Web Server Node:** This node hosts the web application's front-end. It can be labeled as "Web Server" and can be represented as a 3D cube.
- **Application Server Node:** Another node, labeled "Application Server," hosts the back-end logic, APIs, and business logic of the Notebook application.
- **Database Server Node:** A separate node for the database, labeled "Database Server," where all the data (users, tasks, lists) is stored.

Artifacts:

- **Web Application Artifact:** Deployed on the Web Server Node, representing the user interface of the Notebook. This could be a set of HTML, CSS, and JavaScript files.
- **API and Backend Logic Artifact:** Deployed on the Application Server Node, possibly comprising compiled code in the form of a .jar or .war file (if Java is used), or another appropriate format depending on the technology stack.
- **Database Artifact:** Deployed on the Database Server Node, including schema definitions and stored data files (like .sql, .db files).

Associations:

- **Network Connections:** Illustrated with lines or arrows showing the connections between the Web Server and Application Server, and between the Application Server and Database Server.
- **Client Connections:** Representing how users access the system, with arrows from a client device icon (like a laptop or smartphone) to the Web Server.

External Integrations (if applicable):

- **Cloud Services:** If the system uses cloud-based services (like AWS, Azure), these can be represented as separate nodes or labeled within the existing nodes.
- **Third-Party Services:** Nodes or icons representing any external APIs or services the system integrates with, like email servers for notifications.

Security and Load Balancers (optional):

- **Firewall Icon:** Representing security measures around the Web Server and Application Server.
- **Load Balancer:** If the system is designed for high availability and traffic, a load balancer can be shown in front of the Web and Application Servers.

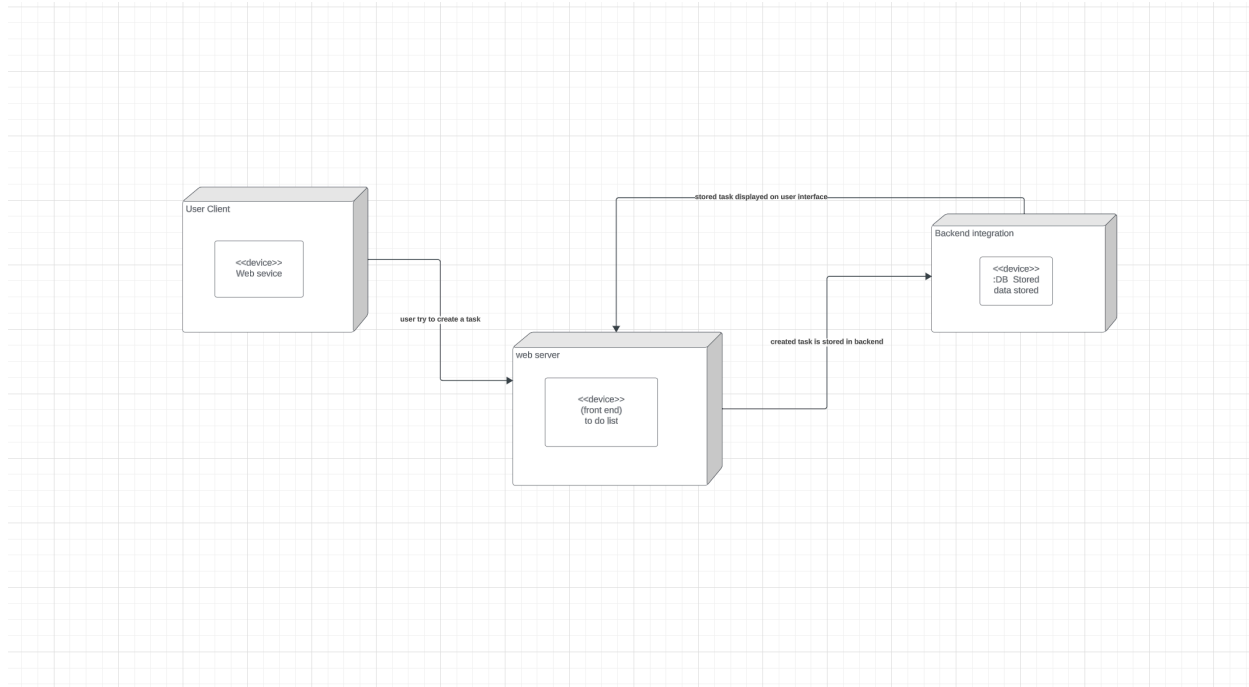


Fig.07. Deployment Diagram

9. Conclusion:

Our Notebook project represents a comprehensive approach to task management, blending modern software engineering principles with user-centric design. Throughout its development, we emphasized functionality, usability, and reliability, ensuring that the application not only meets but exceeds user expectations.

Achievements:

- **Robust System Architecture:** We successfully implemented a scalable and secure architecture, ensuring seamless performance and data integrity.
- **Intuitive User Interface:** The application's user-friendly interface facilitates easy navigation and task management, enhancing the overall user experience.
- **Adaptive Functionality:** Our Notebook application is equipped with features like task prioritization, deadlines, reminders, and search functionality, catering to diverse user needs.

Challenges and Learnings:

The journey was not without challenges. We encountered issues related to cross-platform compatibility and integrating third-party services. However, these challenges were invaluable learning opportunities, allowing us to improve our problem-solving skills and deepen our technical knowledge.

Future Directions:

Looking forward, there is potential for further enhancements:

- **AI-Driven Features:** Implementing AI to suggest task prioritization and deadlines based on user behavior.
- **Enhanced Integration:** Expanding integration capabilities with other productivity tools and platforms.
- **Customization and Personalization:** Offering more personalized experiences based on user preferences and usage patterns.

Closing Thoughts:

This project has been a testament to the power of effective software engineering practices, teamwork, and a user-centered approach. It showcases how technical expertise, when aligned with user needs, can create a product that is not only functional but also impactful in daily life.

In conclusion, our Notebook project stands as a robust, efficient, and user-friendly application, embodying the best practices of software engineering. It serves as a benchmark for future projects, inspiring continued innovation and excellence in the field.