Major Project Report

On

CODE SYNC

Submitted by

Chinmayee Mohanty (2305348011)

Under the esteemed guidance of

Prof. Sumant Sekhar Mohanty

Asst. Professor

Department of Master of Computer Application

In partial fulfilment for the award of the degree of

Master of Computer Application

Gandhi Institute of Excellent Technocrats, Ghangapatna, Bhubaneswar

Biju Patnaik University of Technology, Odisha

2023-2025

BIJU PATNAIK UNIVERSITY OF TECHNOLOGY, ODISHA

CERTIFICATE

*This is to certify that the project report entitled **"CODE SYNC"** is a benefited record of project work carried out by **Chinmayee Mohanty** under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree or diploma.*

Prof. Sumant Sekhar Mohanty
Project Guide
Asst. Professor, Dept. of MCA

Prof. Swarupa Arjya
Head of the Department
Master of Computer application

Department of Master of Computer Application
Gandhi Institute of Excellent Technocrats
Ghangapatna, Khurda, Bhubaneswar
www.gietbbsr.edu.in

DECLARATION

I declare that this project report titled **CodeSync** submitted in partial fulfillment of the degree of Master of Computer Application is a record of original work carried out by me under the supervision of **Prof. Sumanta Sekhar Mohanty**, Asst. Professor at Department of MCA and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Bhubaneswar
Date:17.05.2025

Chinmayee Mohanty (2305348011)

CERTIFICATE OF EVALUATION

COLLEGE NAME: GANDHI INSTITUTE OF EXCELLENT TECHNOCRATS
BRANCH: MASTER OF COMPUTER APPLICATION
SEMESTER: 4th

SL. No.	Registration No.	Name of the Student	Title Of the Project
1	2305348011	Chinmayee Mohanty	Code Sync

The report of the project is submitted by the above student in partial fulfilment for the awards of Degree of Master of Computer Application, Biju Patnaik University of Technology are evaluated and confirmed to the reports of the work done.

Submitted for the University Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behaviour and acts during the course of study.

I also take this opportunity to express a deep sense of gratitude to **Er. Rama Narayan Sabat**, Vice-Chairman**, Prof. (Dr.) Subhrajit Pradhan,** Principal, GIET**,** Ghangapatana, Bhubaneswar, for his cordial support, valuable information and guidance, which helped me in completing this task through various stages.

I wish to express my profound and sincere gratitude to my project guide **Prof. Sumant Sekhar Mohanty,** Assistant Professor, Department of Computer Science & Engineering, who guided me into the intricacies of this project nonchalantly with matchless magnanimity.

I am thankful to **Prof. Swarupa Arjya**, Head of the Master of Computer Application for his support, cooperation and motivation provided to me during the training for constant inspiration, presence and blessings.

I also extend my sincere appreciation to Faculty members are provided valuable suggestions and precious time in accomplishing my major project report.

Lastly, I would like to thank the almighty and my parents for their moral support and friends with whom I shared my day-to-day experiences and received many suggestions those improved the quality of work.

Chinmayee Mohanty (2305348011)

ABSTRACT

Code Sync is a web-based platform offering real-time code execution across multiple popular programming languages including JavaScript, Python, Java, C, Dart, HTML, and CSS. Beyond traditional coding capabilities, Code Sync incorporates two advanced assistive technologies: voice-to-text code conversion and image-to-text extraction. These features enhance accessibility and convenience by enabling users to dictate code verbally and extract code from images such as screenshots or handwritten notes, making the platform inclusive for diverse users including students, educators, and developers. The platform allows users to write, execute, and download code directly within the browser, eliminating the need for any local setup or configuration. Its backend securely handles code compilation and execution through language-specific scripts, ensuring safe and efficient processing of user-submitted code. The frontend provides a clean, responsive, and intuitive user interface that performs smoothly on both desktop and mobile devices, facilitating an engaging coding experience. By combining real-time code execution with voice and image-based input methods, Code Sync streamlines coding workflows and fosters a more accessible learning environment. This innovative approach promotes programming engagement and supports diverse learning styles by integrating assistive technologies seamlessly into the coding process. Overall, Code Sync offers a powerful, user-friendly solution designed to improve the coding experience through inclusive and efficient technology.

Keywords: Real-time code execution, Multi-language support, Voice-to-text conversion, Image-to-text extraction, Web-based IDE, Responsive user interface, Code compilation, Browser-based coding

CONTENTS

CHAPTER 1

INTRODUCTION

1.1. PURPOSE

The main purpose of the Code Sync project is to develop a web-based platform that enables users to write, execute, and download code in real time across multiple programming languages such as JavaScript, Python, Java, C, Dart, HTML, and CSS. Unlike traditional IDEs that require installation or local setup, Code Sync operates entirely in the browser, offering a zero-configuration experience. It is intentionally designed as a non-collaborative platform, focusing on individual users who prefer a private, distraction-free coding environment.

In addition to conventional code editors, Code Sync introduces two assistive technologies:

- Voice-to-text, which allows users to dictate code through speech, making coding accessible to people with motor impairments or those who find typing challenging.
- Image-to-text (OCR), which can extract code from images, such as handwritten notes or printed textbook pages.

The platform aims to support diverse user groups—students, educators, differently-abled individuals, and beginner programmers—by offering tools that simplify the process of coding and reduce the technical barriers. Through these features, Code Sync promotes inclusivity, enhances productivity, and fosters an engaging and modern learning environment for anyone wanting to practice or test code efficiently.

1.2. Project Scope

The scope of the Code Sync project is to build a cross-platform, browser-based coding environment that eliminates the need for any software installation or complex configuration. The application aims to deliver a fully functional online IDE capable of real-time code writing, execution, and downloading, while enhancing user interaction through voice-to-code and image-to-code (OCR) functionalities. The system is optimized for a broad audience including students, educators, freelancers, and differently-abled users.

The key elements covered under the project scope are:

- Multi-language support: Code Sync supports various popular programming languages like JavaScript, Python, Java, C, Dart, HTML, and CSS, catering to both frontend and backend development learning needs.

- Real-time Code Execution: Users can write and execute code directly in the browser. Code is sent to a secure backend, where it is compiled/interpreted in isolated environments using language-specific scripts.
- Voice-to-Text Coding: With integrated voice recognition, users can dictate their code verbally, making the process more accessible for those with physical limitations or learning difficulties.
- Image-to-Text (OCR): Users can upload images containing code, which are processed using Optical Character Recognition to extract editable text, helping in situations where typing is slow or manual transcription is error-prone.
- Code Downloading: After writing and executing their code, users can download their work in the selected programming language format, enabling offline access or submission.
- Responsive UI Design: Code Sync features a mobile-first, responsive interface, ensuring consistent performance across desktops, tablets, and smartphones. This enhances learning on the go.

This wide scope of functionalities enables Code Sync to be more than just a web IDE—it becomes a learning companion, productivity booster, and accessibility-enhancing platform that supports diverse use cases across the coding spectrum.

Problem Definition

Most Existing online IDEs often fail to meet the needs of individual users and learners. The major problems include:

- Distraction from Unnecessary Collaboration:
  Most platforms focus on collaborative features, which can overwhelm individuals seeking a quiet, focused coding environment.
- Lack of Accessibility Features:
  Users with physical disabilities or motor impairments struggle due to the absence of assistive tools like voice input and OCR.
- Complicated Setup Requirements:
  Beginners often face difficulties installing compilers or interpreters for different programming languages.
- Limited Multi-language Support:
  Many platforms support only a few languages, restricting flexibility for learners and developers.
- No Support for Visual or Handwritten Code Input:
  There's often no way to convert code from images, notes, or textbooks into editable format.

CHAPTER 2

LITERATURE SURVEY

2.1 SYSTEM REVIEW

Several web-based IDEs such as Replit, JSFiddle, CodePen, and LeetCode are widely used for various purposes like collaborative coding, competitive programming, or front-end development. However, from the perspective of usability for solo developers, learners, and accessibility-centric users, they lack in several areas. Code Sync is designed to overcome these limitations. Here's a point-wise review of current systems:

- Lack of Assistive Features: Most platforms do not integrate assistive technologies like voice input (Speech Recognition) or Optical Character Recognition (OCR) for image-to-code conversion.
- Focus on Collaboration: Platforms such as Replit and CodePen promote real-time collaboration, which might distract individual users who prefer a quiet, private environment.
- Beginner Unfriendly: Many existing IDEs are loaded with advanced features or require account setup, which can be overwhelming for new learners.
- Limited Language Flexibility: Some platforms only support frontend code (like HTML/CSS/JS) or a specific set of backend languages.

Code Sync Addresses These Issues By:

- Offering a private, distraction-free interface for solo users.
- Providing voice-to-text and OCR-based input for inclusive development.
- Supporting multiple programming languages in one place.
- Ensuring simplicity without sacrificing powerful features.

2.2 TECHNOLOGY USED

Frontend Technologies:

- HTML5 & CSS3: Structure and design layout.
- JavaScript (ES6+): Dynamic functionality.
- React.js: For building a responsive, Single Page Application (SPA) interface with reusable components.

Backend Technologies:

- Node.js: Runtime for server-side logic.
- Express.js: Framework for handling API requests and routing.
- Child Process Module: Securely executes user-submitted code in isolated environments.
- Voice-to-Text (Speech Input):
- Web Speech API / SpeechRecognition API: Enables users to dictate code directly using a microphone.
- Image-to-Text (OCR):
- Tesseract.js: A JavaScript OCR engine used to extract code from screenshots, scanned notes, or textbook images.
- Security & Isolation:
- Sandboxed Execution: Executes user code in isolated environments.
- Memory & Time Limits: Prevents abuse or infinite loops.
- Permission Restrictions: Disallows system-level access or I/O operations.

Database :

- MongoDB: used for user management, project persistence, and session handling.

CHAPTER 3

Requirement Analysis

3.1 Programming Language

Languages Used in Development:

- Frontend Logic: JavaScript (React)
- Backend Logic: JavaScript (Node.js + Express)

Languages Supported for Code Compilation:

- JavaScript
- Python
- Java
- C
- Dart
- HTML + CSS (rendered in browser)

Key Points:

- Multi-language support enables versatile use-cases.
- Each language runs in a secure, sandboxed environment.
- Input/output for code execution is handled in real-time.
- Code can be downloaded as a file in the appropriate language format.

3.2 Operating System

Cross-Platform Compatibility:

- Code Sync is entirely web-based, meaning it works across all major operating systems without requiring any installations.

Supported Operating Systems:

- Windows, macOS, Linux, Android, iOS

- Users can access Code Sync from any device that supports a modern web browser.

Browser Support:

- Google Chrome, Mozilla Firefox, Microsoft Edge, Safari
    - The platform is optimized for the latest browser versions and provides a responsive interface for both desktop and mobile screens.

Key Points:

- No Installation Required: Runs directly from the browser.
- Responsive Design: Adapts layout automatically for screen sizes (PCs, tablets, mobiles).

3.3 Hardware & Software Requirement

Hardware Requirements:-

To use Code Sync, users do not need any special or expensive hardware. The platform is designed to work on basic devices that have internet access and a modern web browser. The minimum hardware requirements are:

- Device:
    - Any modern desktop, laptop, tablet, or smartphone. Desktops/laptops are preferred for better experience.
- Processor:
    - Minimum 1.0 GHz. No need for high-end CPUs or GPUs.
- RAM:
    - Minimum 2 GB. 4 GB or more recommended for better performance.
- Storage Space:
    - At least 500 MB of free space for saving downloaded code files and other resources.
- Internet Connection:
    - Stable broadband or 4G connection required for real-time syncing.
- Webcam & Microphone:
    - Not required currently. Optional for future features like video or voice communication.

Software Requirements:

Since **Code Sync** is a web-based application, it has very simple software requirements. Users do not need to install any heavy software or special programs on their devices. Everything runs directly through a web browser with an internet connection.

The main software requirements are:

Operating System:

Code Sync works across all major operating systems because it is accessed through a browser. Supported operating systems include:

Windows (Windows 7, 8, 10, 11 and newer versions)

MacOS (MacBooks and iMacs)

Linux (Ubuntu, Fedora, and other distributions)

Android (Smartphones and Tablets)

IOS (iPhones and iPads)

There is no restriction on the type of OS because the application is built using responsive design and modern web technologies.

Web Browser:

To use **Code Sync**, users must have a modern web browser that supports HTML5, CSS3, and JavaScript. Supported browsers include:

Google Chrome (Recommended for the best performance and full compatibility)

Mozilla Firefox (Good performance and privacy features)

Microsoft Edge (Works well on Windows devices)

Safari (for Apple devices like MacBooks, iPhones, and iPads)

It is important to have the latest version of the browser to ensure full functionality, speed, and security.

Additional Browser Features:

JavaScript must be enabled in the browser settings.

Cookies should be allowed to maintain user sessions (login, project storage).

Browser pop-ups should not be blocked for essential notifications.

No Installation Needed:

Code Sync does not require any separate software installation like desktop applications. Everything runs through the browser itself, making it light, fast, and easy to access from anywhere at any time.

Developer Requirements (for building or improving Code Sync):

For developers who want to work on or extend the Code Sync platform, the following tools are recommended:

Visual Studio Code (or any preferred code editor) for writing and editing code.

Node.js and npm (Node Package Manager) for managing libraries if local testing or modifications are needed.

Git for version control during project development.

CHAPTER 4

System Design

4.1 Use Case Diagram

The **Use Case Diagram** for **Code Sync** shows how users interact with the system and what main actions (use cases) they can perform on the platform. It helps in understanding the system's overall functionality from the user's point of view.

In **Code Sync**, the main actor is the **User** (which could be a student, developer, or team member). The User interacts with the system to perform different tasks such as creating a new project room, joining an existing room, editing code, uploading files, chatting with teammates, and logging out after the session.

Actors:

User (Primary Actor):

The user is the central participant of the CodeSync platform. This includes:

- **Students**, who use the platform to learn programming, run code snippets, and understand output behavior in real-time.
- **Educators**, who demonstrate code examples, assignments, or project snippets through voice or image inputs during teaching.
- **Professional Developers**, who may quickly prototype, test, and save or share code without needing to set up a local development environment.

These users interact with Code Sync via a web browser, utilizing both the traditional and assistive features to streamline their coding workflows.

System (Secondary Actor):

This represents the backend engine of CodeSync, responsible for:

- Receiving and securely executing code in various languages (JavaScript, Python, Java, C, etc.).
- Handling voice input by converting dictated speech into syntactically correct code.
- Processing image input, such as screenshots or handwritten notes, and extracting readable code using OCR (Optical Character Recognition).
- Returning output or feedback based on execution, validation, or error handling.

Main Use Cases:

Register/Login/Logout:

Allows users to create an account, log in securely, and maintain session-based access. This helps personalize the experience, track activity, and secure user data.

Write Code:

Users can write code directly in the browser using a responsive, multi-language-supported editor. Syntax highlighting and formatting enhance the coding experience.

Execute Code:

The code written by the user is sent to the backend for safe execution. The result—either output or errors—is displayed instantly. This real-time feedback improves learning and testing.

Copy Code:

A one-click option allows users to copy code snippets to the clipboard for use in other platforms like IDEs, email, or chat. It improves productivity and facilitates faster sharing.

Download Code:

Users can download their code files in respective formats (e.g., .py, .js, .c) for offline use, submission, or archival purposes.

Voice-to-Text Input:

This feature allows users to dictate code verbally. The system interprets spoken programming constructs, keywords, and syntax into actual code lines. Useful for users with disabilities or those multitasking.

Image-to-Text Extraction:

Users can upload images containing code—whether printed, handwritten, or screenshot-based. The system extracts the code and populates the editor, saving time and manual transcription effort.

Use Case Diagram Explanation:

The Use Case Diagram visualizes how the user interacts with the different functionalities provided by the CodeSync system.

- The User initiates use cases like writing, copying, executing, downloading, or entering code through voice or image.
- The System executes the backend logic, translating non-text inputs (voice/image) into code and compiling/executing code based on the selected language.
- Arrows represent the interactions between user and system functionalities, showing the flow of actions like input, processing, and response.
- It distinguishes between direct user actions (e.g., writing or executing code) and automated or system-driven operations (e.g., converting voice to text or extracting text from image).

This diagram helps developers, stakeholders, and educators understand the workflow and scope of the application from a user-centric perspective.

Importance of Use Case Diagram:

- **Enhancing Accessibility:**

By including voice and image recognition, CodeSync caters to users who may have physical or situational limitations. It enables people with motor impairments, vision problems, or learning difficulties to code more comfortably.

- **Boosting Productivity:**

Features like real-time code execution, one-click copy, and downloads streamline the development cycle. Users can quickly test, revise, and export code without switching environments.

- **Facilitating Education and Learning:**

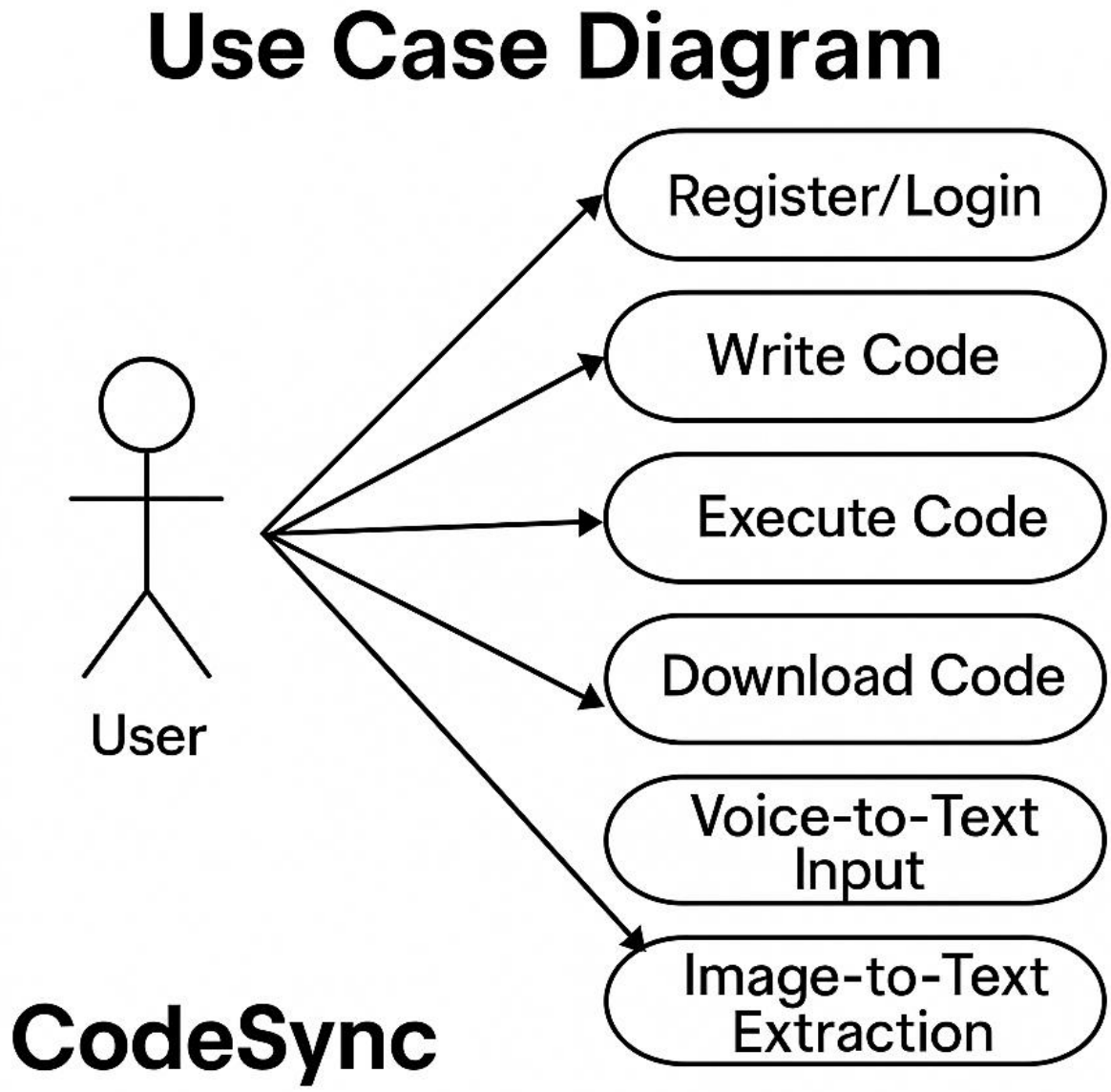
CodeSync is ideal for coding classes, online workshops, and collaborative learning. Instructors can demonstrate code quickly through speech or screenshots, while students can instantly test what they learn.

- **Reducing Setup Complexity:**

There's no need for installations or complex configurations. The user can simply open the platform in a browser and start coding across multiple languages, making it ideal for both rapid prototyping and educational purposes.

- **Supporting Multimodal Inputs:**

Unlike conventional editors, CodeSync breaks barriers by letting users provide input through typing, speaking, or uploading images—making it a truly **inclusive and modern development environment**



4.2 Activity Diagram

An **Activity Diagram** shows the flow of actions in a system from the beginning to the end. It explains how users move systematic through the system to complete their tasks. In the **Code Sync** project, the Activity Diagram represents the flow of how a user interacts with the platform — starting from login then select a programming language from the dashboard. In the editor, they write and run code with real-time output, using voice-to-text and image-to-text featuresand users can copy, download, or log out securely.

An **Activity Diagram** is a type of flowchart that represents the sequence of activities or operations in a system. It shows how the user moves through the system, step-by-step, and what actions they perform at each stage. For the **Code Sync** project, the Activity Diagram explains the process from starting the platform to ending a session, helping developers and users clearly understand the flow of actions.

CodeSync's workflow begins when users visit the website and are prompted to either register a new account or log in with existing credentials. New users provide basic information to create an account, while returning users simply enter their login details. Upon successful authentication, users are redirected to the main dashboard, ensuring a secure and personalized environment for coding activities.

Once logged in, users encounter a clean and responsive dashboard that supports both desktop and mobile devices. From here, users choose their preferred programming language, such as JavaScript, Python, Java, C, HTML, or CSS. Selecting a language automatically takes the user to a dedicated code editor page tailored for that language, streamlining the transition from navigation to coding.

Inside the editor, users can write, run, and test code with real-time output displayed instantly. CodeSync supports multiple languages and ensures secure backend compilation and execution. The editor also features advanced assistive tools, including voice-to-text conversion for dictating code and image-to-text extraction using OCR technology to transform code from screenshots or handwritten notes into editable text, enhancing accessibility and convenience.

Users can easily copy or download their code for offline use. The platform automatically saves progress to the cloud via Firebase, protecting work from accidental loss. After completing their coding session, users can navigate back to the dashboard or access their profile to log out securely, concluding their session. This structured workflow combines traditional coding functions with innovative features to provide an inclusive, efficient coding experience.

Activity Flow for Code Sync:

Start

↓

User Opens CodeSync Website

↓

Login / Register

├── New User → Register

└── Existing User → Login

↓

Navigate to Dashboard

↓

Choose Programming Language

↓

Redirect to Code Editor

├── Write and Run Code (Multi-language support)

├── Voice-to-Text Code Input

├── Image-to-Text Code Extraction (From screenshot or handwritten notes)

├── Copy or Download Code

↓

Return to Dashboard

↓

Profile → Logout

↓

End

An **Activity Diagram** is a type of UML (Unified Modelling Language) diagram that shows the workflow of actions and activities in a system.

It represents the order of operations performed by the user or system to complete a particular task. In **Code Sync**, the Activity Diagram explains the systematic process users follow to collaborate on coding projects using the platform.

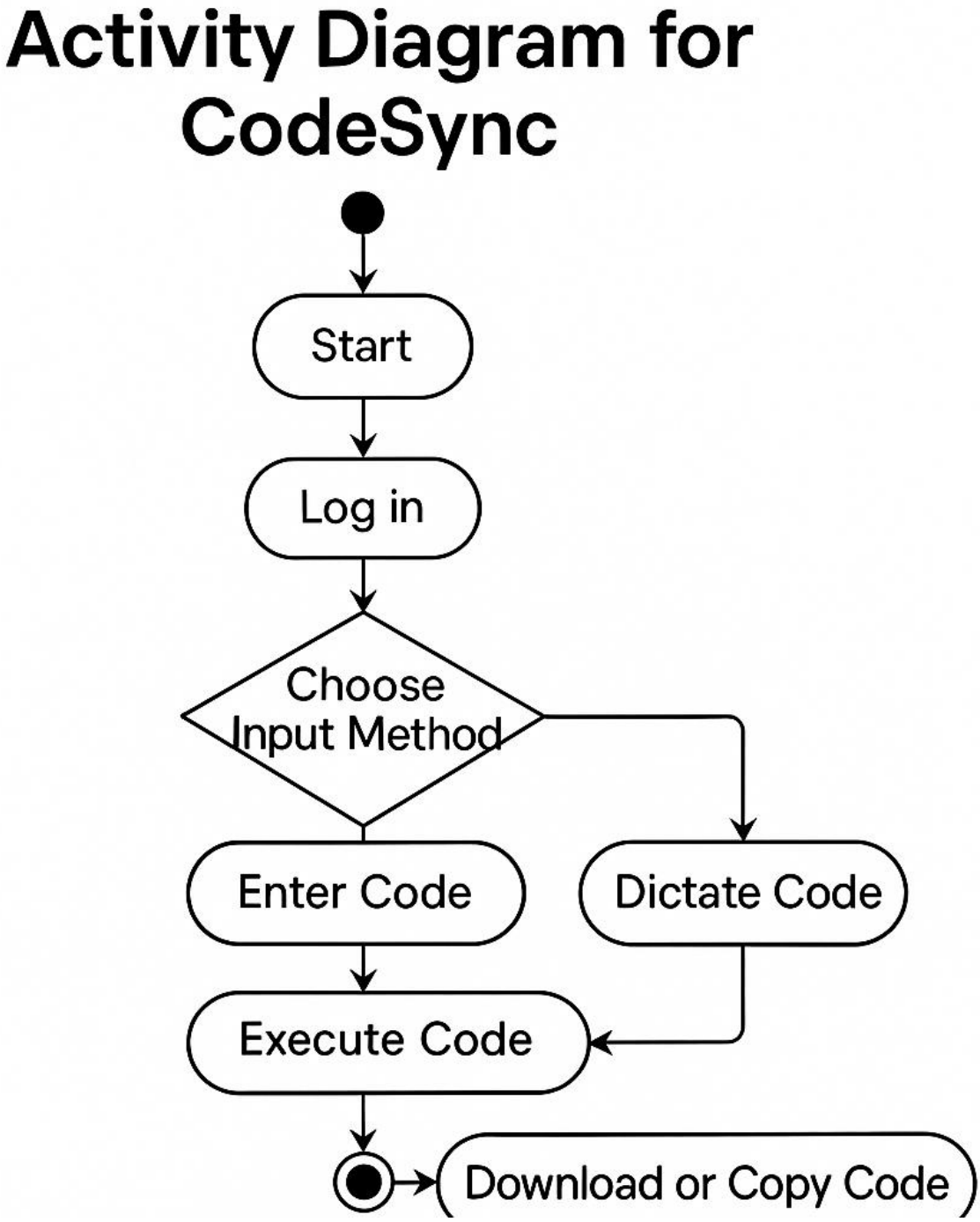
The diagram begins when the user accesses the **Code Sync** website. The first activity is **Register/Login**. If the user already has an account, they log in; if not, they must register a new account.

Purpose of Activity Diagram in Code Sync:

An Activity Diagram is a UML (Unified Modeling Language) diagram that visualizes the sequence of activities a user performs within a system. In the CodeSync project, the activity diagram maps the complete user journey, from accessing the platform to performing advanced actions like real-time code execution, voice-to-text conversion, image-to-text extraction, and finally logging out.

This diagram helps:

- Clearly visualize the workflow.
- Identify opportunities for feature enhancements.
- Assist testers in verifying flow integrity.
- Enhance the user experience through structured design.



4.3 E-R Diagram

An **Entity-Relationship (ER) Diagram** is a visual tool used to model the structure of a database. It shows the important data objects (**entities**), their details (**attributes**), and how they are connected (**relationships**) inside a system.

For the **Code Sync** project, the ER Diagram explains how users, rooms, files, and chat messages are linked within the platform's database.

In Code Sync, the main entity is the **User**, which stores details like User ID, Name, Email, and Password. Each user can either **create** or **join** a coding room. The **Room** entity represents a live coding environment where collaboration happens. It includes Room ID, Room Name, and Creation Time.

Another important entity is **File**. Users inside a room can upload project files. Each file has attributes like File ID, File Name, File URL (location where the file is stored), and Upload Time.

Similarly, the **Message** entity handles real-time chat communication between users. It includes Message ID, Sender ID, Room ID (where the message is posted), Content (the text), and Timestamp.

It shows the important entities (objects), the attributes of those entities, and the relationships between them. In the **Code Sync** project, the ER Diagram explains how users, coding rooms, files, and chat messages are related to each other inside the system's database.

The main entities in the Code Sync system are:

- **User:** Represents people who use the platform. Important attributes include User ID, Name, Email, and Password.
- **Room:** Represents a coding room created for collaboration. Attributes include Room ID, Room Name, and Created Time.
- **File:** Represents project files uploaded by users. Attributes include File ID, File Name, File URL, and Upload Time.
- **Message:** Represents chat messages between users. Attributes include Message ID, Content, Sender ID, and Timestamp.

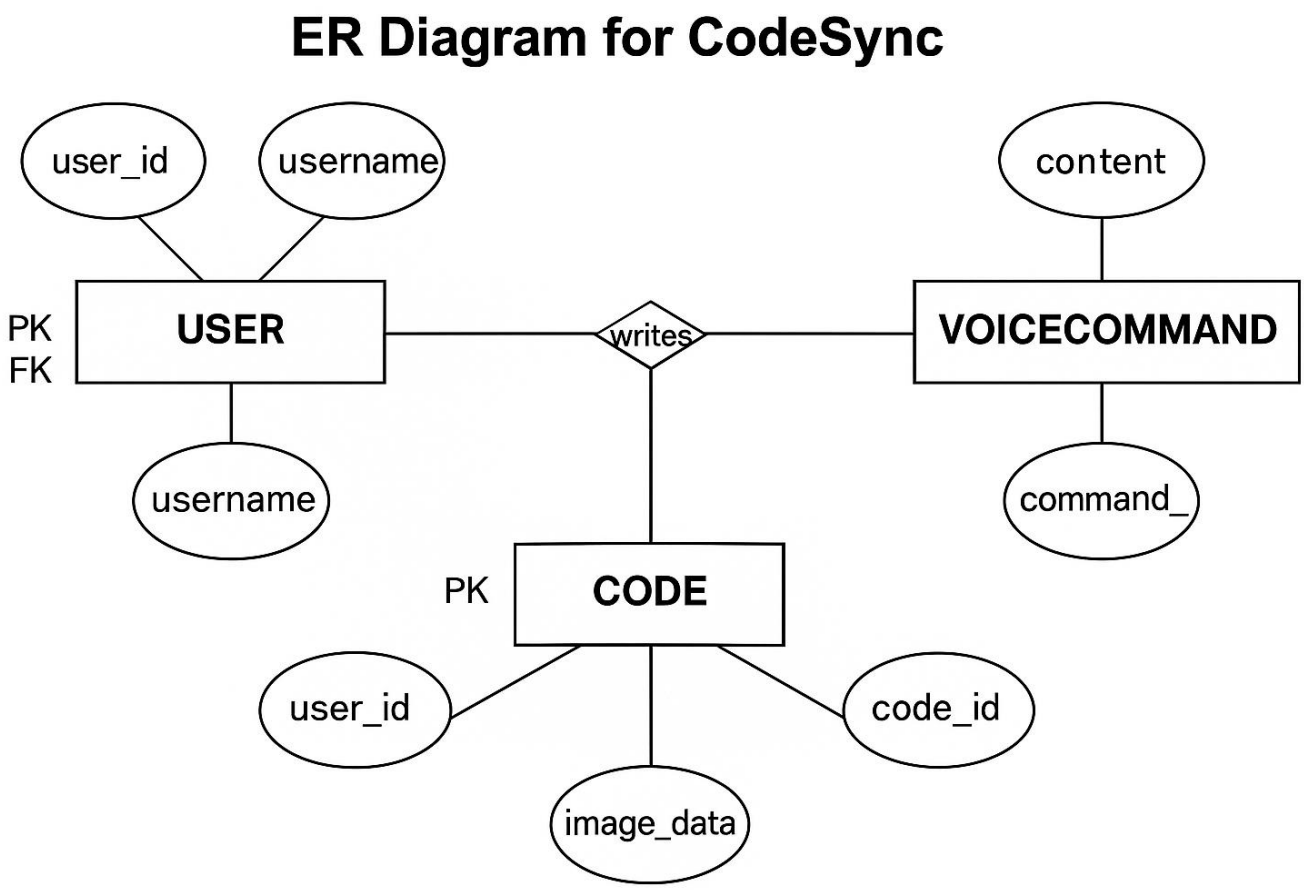
This ER Diagram is very important because it ensures that the database is well organized, reduces duplication of data, improves data integrity, and makes it easy for the system to handle many users and projects at the same time. It acts like a blueprint for database developers to build and manage the backend properly.

Relationships:

- A **User** can create or join multiple **Rooms**.
- A **Room** can have multiple **Users** participating.
- A **User** can upload multiple **Files** inside a **Room**.
- A **User** can send multiple **Messages** inside a **Room**.

The ER Diagram clearly defines how the data is stored, linked, and managed inside the database, ensuring that all user actions (like coding, chatting, and uploading files) are properly saved and organized.

It also helps developers design an efficient database system that can handle multiple users and activities at the same time without confusion or data loss.



[E-R Diagram of Code Sync]

CHAPTER 5

System Testing

5.1 Test Cases & Test Result

Testing is one of the most important phases in the software development life cycle. It is the process of checking and validating whether the developed system meets the specified requirements and works correctly. The main aim of testing is to find errors, bugs, or missing functionalities before the system is released to the users. Good testing ensures that the final product is reliable, efficient, secure, and user-friendly.

For the **Code Sync** project, testing played a very important role. Since Code Sync is a real-time collaborative coding platform, it involves multiple users working together at the same time, uploading files, sending messages, and editing code simultaneously. Therefore, every feature had to be tested carefully to ensure it worked correctly under different conditions such as different devices, browsers, and internet speeds.

Manual testing was used in this project because it allowed direct checking of the system's functionality by acting as a real user. Features like user registration, login, room creation, joining rooms, real-time code editing, file uploading, and chat communication were all tested one by one. Different test cases were prepared to check how the system reacts to correct inputs as well as incorrect inputs.

Testing helps in finding hidden issues early, reduces future maintenance costs, and improves the overall quality of the system. Through proper testing, **Code Sync** was made stable and ready for real-world use. Testing also builds confidence that the application will perform well when multiple users collaborate from different locations.

For the **Code Sync** project, we carried out **manual testing** on all major features including registration, login, room creation, code editing, file uploading, chatting, and logging out.

Thus, testing ensures that the Code Sync platform is error-free, smooth, and delivers a great user experience.

Test Strategy

A Test Strategy defines the overall approach to testing the system. It describes how the testing will be carried out, what types of testing will be done, which tools or environments will be used, and the goals of the testing process.

For the **Code Sync** project, the testing strategy focused on **manual functional testing**. The main goal was to ensure that all features, such as user login, room creation, code collaboration, file uploading, and real-time chat, work correctly and smoothly.

Testing was conducted on different devices like laptops, desktops, and smartphones to ensure cross-device compatibility. Browsers like Google Chrome and Mozilla Firefox were used for browser compatibility testing. Different network conditions (Wi-Fi and 4G) were used to test performance under various speeds.

All critical user actions were tested based on prepared test cases. Positive testing (with correct input) and negative testing (with wrong input) were both done to find any issues. Bugs found during testing were fixed immediately to improve system performance.

The aim of the test strategy was to:

- Ensure the system meets all functional requirements.
- Make the platform stable and user-friendly.
- Identify and fix errors before final deployment.

By following this testing strategy, **Code Sync** was fully checked to guarantee a smooth and reliable collaborative experience for all users.

• **Type of Testing:** Manual Functional Testing

• **Testing Environment:**

- Devices: Laptop (Windows 10), Smartphone (Android 12)
- Browsers: Google Chrome, Mozilla Firefox
- Network: Wi-Fi Broadband and Mobile Hotspot (4G)

• **Goal:** To check that all main user actions work properly under normal conditions.

Type of Testing:

Manual functional testing was used for **Code Sync**. Each feature, like login, room creation, code editing, file uploading, and chat, was tested by manually interacting with the system to ensure everything worked correctly.

• **Testing Environment:**

The system was tested on different devices, including laptops, desktops, and smartphones. Popular browsers

like **Google Chrome** and **Mozilla Firefox** were used to check browser compatibility. Testing was done over Wi-Fi and 4G mobile networks to observe system behaviour under different internet speeds.

- **Test Cases Preparation:**

Detailed test cases were created for each feature. These test cases included both valid (positive) and invalid (negative) inputs. The goal was to check how the system behaves with correct actions and how it handles errors or wrong inputs.

- **Bug Reporting and Fixing:**

During testing, if any bugs or issues were found (such as minor display errors or chat delays), they were immediately reported to the developer. After fixing, the affected features were retested to confirm that the issues were resolved.

- **Goal of Testing:**

The main goal of testing was to make sure that all functionalities worked as expected, the system was stable, user-friendly, and ready for real users. Testing also helped ensure that **Code Sync** performs well even when multiple users are connected at the same time.

The testing strategy for **Code Sync** involved manual functional testing across all major features like user registration, login, room creation, code editing, file uploading, chat messaging, and logout.

Testing was performed on various devices, including laptops, desktops, and smartphones, using browsers like Google Chrome, Mozilla Firefox, and Microsoft Edge.

Different internet conditions, such as Wi-Fi and 4G networks, were used to check system performance in real-time collaboration. Both positive and negative test cases were prepared to ensure the platform handled correct and incorrect inputs properly.

Real and dummy data were used to test file uploads, chat messaging, and multi-user collaboration. Bugs found during the process were reported, fixed, and retested to ensure high system stability.

Regression testing was also carried out to make sure no new errors appeared after fixes.

The main aim of testing was to confirm that **Code Sync** is reliable, user-friendly, and performs smoothly under different conditions.

Testing ensured that the platform was ready for real-world usage by students, developers, and small teams for collaborative coding.

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Status
TC01	User Registration	Email & Password	Account Created	As Expected	Pass
TC02	User Login	Valid Credentials	Logged into Dashboard	As Expected	Pass
TC03	Choose Language	Language Name	Snippet Created with ID	As Expected	Pass
TC04	Editor Page	Snippet Code	Joined the Snippet	As Expected	Pass
TC05	Code Editing	Typing code	Real-time Updates	As Expected	Pass
TC06	File download	Code/Doc Files	Downloaded Successfully	As Expected	Pass
TC07	Execute	Sending output Message	Execute successfully	As Expected	Pass
TC08	Logout	Clicking Logout	Redirected to Login Page	As Expected	Pass

Test Cases Table

Test Case Table is a simple and organized way to **show all the tests** you performed on your project during the testing phase.

Each **test case** is like a small task where you check if a particular feature (like login, uploading a file,

[TEST CASE TABLE OF CODE SYNC]

Test Execution Process

The **Test Execution Process** is the stage where all the prepared test cases are actually performed on the system to check if it behaves correctly.

In this phase, testers follow the steps mentioned in the test cases, provide the necessary inputs, observe the system's response, and record the results.

For the **Code Sync** project, the test execution process was carried out manually. Each important feature — such as user registration, login, room creation, joining rooms, code editing, file uploading, and chatting — was tested one by one according to the test case table.

Testing was done on different devices like laptops and smartphones, using various browsers (Chrome, Firefox) and different internet networks (Wi-Fi and 4G).

Positive test cases (correct inputs) and negative test cases (wrong inputs) were executed to check both expected behaviour and error handling.

During execution:

- If the actual result matched the expected result, the test case was marked as **Pass**.
- If there was a mismatch or an error, it was marked as **Fail**, and the issue was recorded for fixing.

After fixing any issues, **re-testing** was done to make sure the problem was solved properly. This helped ensure that all parts of **Code Sync** worked smoothly before final deployment.

Every feature of **Code Sync** was tested with different users connected from different devices. Example tests included:

- Logging in with different browsers.
- Creating and joining rooms with different users.
- Uploading different types of files (.txt, .docx, .png).

- Testing chat performance in slow internet conditions.

Additionally, network testing was done by connecting through Wi-Fi and mobile 4G internet to observe the system's behaviour under different speeds. After identifying and fixing any errors, re-testing was performed to confirm that issues were resolved.

In simple words, the Test Execution Process checked that all features worked correctly, and the system was stable and ready for users.

CHAPTER 6

Project Planning

6.1 Timeline & Gantt chart

Timeline

A **project timeline** is a chronological list of tasks, deadlines, and milestones for a project. In the context of **Cody Sync**, the timeline outlines **key development phases**, such as:

1. **Planning**

This is the foundation of the entire project.

- Define the purpose of Cody Sync (e.g., file sharing, code previewing).
- Identify target users and key features.
- Create a basic roadmap.
- Decide on technologies (e.g., Firebase, React, Tailwind)

2. **Design (UI/UX)**

This phase focuses on how Cody Sync will look and feel to users.

- Design wireframes (layout and structure).
- Choose colour schemes, icons, and typography.
- Plan user flows (e.g., upload > share > view).
- Ensure responsive and mobile-first design.

3. **Backend Setup**

This is where the functionality of the app begins to take shape.

- Set up Firebase Authentication for login.
- Configure Fire store or Real-time Database**.**
- Enable Cloud Storage for file uploads.
- Define roles, permissions, and access rules

4. **Frontend Development**

Build the actual interface users will interact with.

- Create pages Login, Dashboard, File Upload, File Viewer, Share modal.
- Integrate with backend (e.g., upload files to Firebase).
- Handle file previews and code rendering (e.g., Prism.js).
- Optimize for low-power devices.

5. **Testing & QA**

Before launching, the app must be tested for bugs, performance, and security**.**

- Test on different devices and browsers.
- Upload and download various file sizes.
- Check share links and permissions.
- Fix UI glitches or crashes.

6. **Deployment**

Time to make Cody Sync live!

- Host the frontend (e.g., on Firebase Hosting).
- Set up the production database and storage.
- Configure domains and SSL.
- Test everything again in live mode.

7. **Post-launch Updates**

After launch, continue to improve and support the app.

- Monitor for bugs or feedback.
- Add new features (e.g., file versioning, collaboration).
- Optimize performance.
- Maintain security and fix vulnerabilities.

Gantt chart

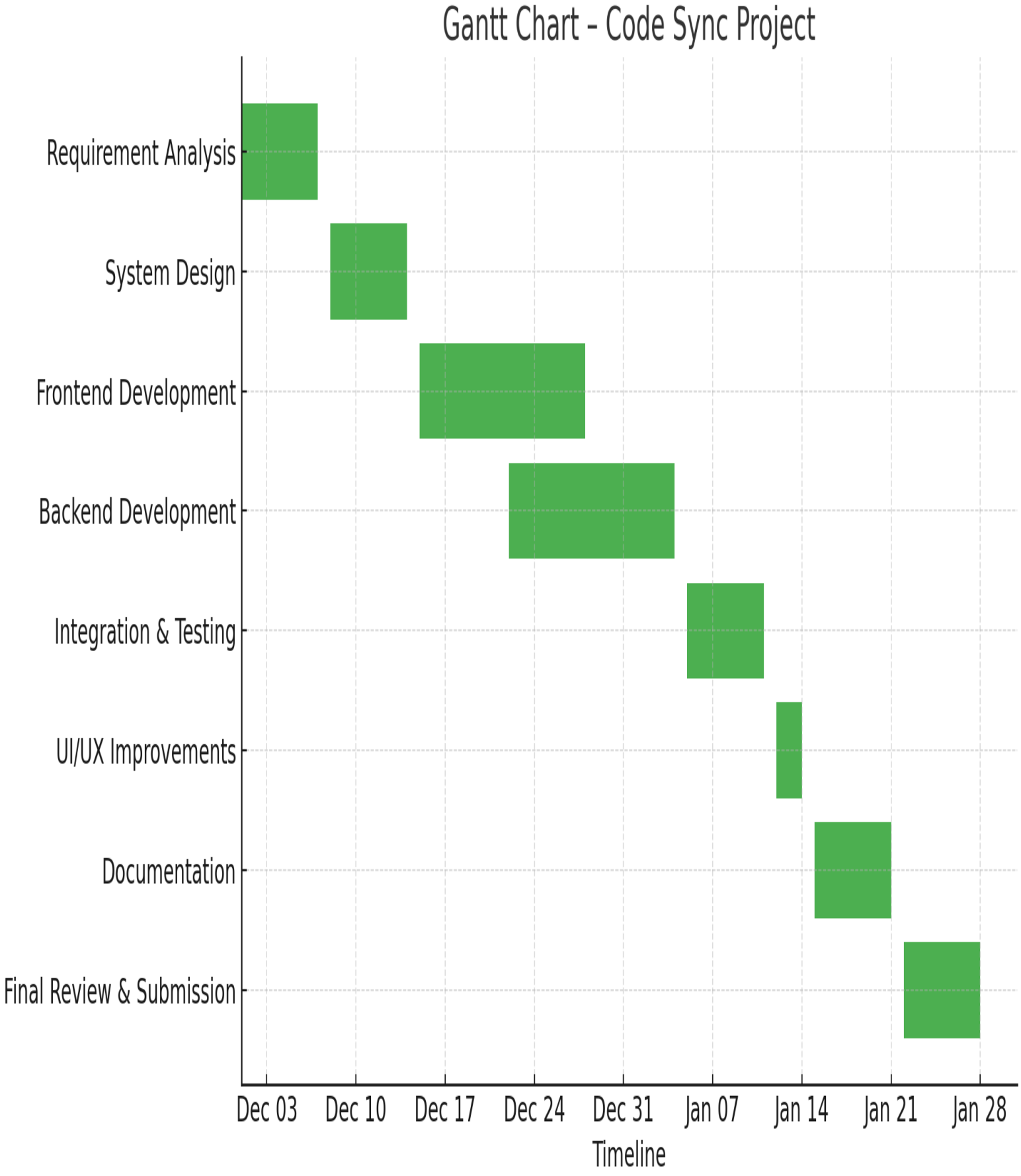
A Gantt chart is a visual project management tool that displays tasks along a timeline, helping teams track what needs to be done, when, and by whom.

Each task is represented as a horizontal bar, showing its start date, end date, and duration. Gantt charts make it easy to see task dependencies, overlaps, and project progress at a glance.

For a project like Cody Sync, it ensures all development stages—from planning to deployment—are organized, timely, and well coordinated.

- Tasks as horizontal bars
- Start and end dates
- Task duration
- Overlaps or dependencies between tasks

Task	Duration	Start	End	Depends
Requirement Gathering	4 days	Mar 15	Mar 18	
UI/UX Design	5 days	Mar 19	Mar 24	Requirements
Firebase Setup (Auth + DB)	3 days	Mar 22	Mar 24	UI/UX partial
File Upload System	4 days	Mar 25	Mar 28	Firebase Setup
Real-time File Viewer	5 days	Mar 29	Apr 4	File Upload
Sharing & Permissions	3 days	Apr 5	Apr 8	Viewer
Testing & Bug Fixes	4 days	Apr 10	Apr 14	All dev tasks
Deployment & Launch	10 days	Apr 17	May 27	All dev tasks


Gantt Chart – Code Sync Project
Requirement Analysis
System Design
Frontend Development
Backend Development
Integration & Testing
UI/UX Improvements
Documentation
Final Review & Submission
Dec 03
Dec 10
Dec 17
Dec 24
Dec 31
Jan 07
Jan 14
Jan 21
Jan 28
Timeline

CHAPTER 7

Implementation

7.1 Work Flow

Introduction

CodeSync is a real-time, multi-language online code editor that enhances the programming experience through features like voice-to-text and image-to-text code input. Built with a responsive design, it allows users to write, execute, and download code directly from the browser without any additional setup. The platform is designed for students, developers, and educators, focusing on accessibility, speed, and ease of use.

1. User Access & Authentication

When users visit the CodeSync platform:

- New Users must register by providing basic details such as name, email, and password.
- Existing Users log in using their credentials.

This process ensures secure access and personalized sessions. Authentication is handled with proper backend validation, ensuring account safety and role-based access where needed.

2. Lightweight Dashboard Interface

After successful authentication, users land on the main dashboard:

- Clean and responsive design adaptable to both desktop and mobile.
- Allows users to view their profile or navigate directly to coding features.
- Provides options for selecting different supported programming languages.

This dashboard acts as the central hub from where users can control their entire session.

3. Choose Programming Language

Users select the desired programming language from available options:

- Supported languages include JavaScript, Python, Java, C, HTML, and CSS.
- Upon selection, users are immediately directed to the appropriate code editor tailored for the chosen language.

4. Redirect to Code Editor

This is the heart of CodeSync, where the actual coding happens:

- Code Writing & Execution: Users write code in the editor and see real-time outputs.
- Voice-to-Text: Converts spoken code instructions into actual code, ideal for accessibility or hands-free coding.
- Image-to-Text: Uses OCR to extract code from screenshots or handwritten notes.

- Multi-Language Support: The editor dynamically adjusts to the selected language, ensuring syntax highlighting and backend execution.

5. File Download & Open

Once the code is written and tested:

- Users can copy the code for sharing or pasting elsewhere.
- Or download the code file for offline use or future reference.
- The system ensures smooth code management without needing additional extensions or tools.

6. Logout

When the user finishes:

- They return to the dashboard or visit the profile section.
- From here, users can securely log out, ending the session and preserving data integrity.

7.2 Source Code

Front end: -

(Home Page)

```
import React from "react";
// import ImageCoding from '../../assets/JavaScript frameworks-rafiki.svg'
import { NavLink } from "react-router-dom";
function CodingPage(props) {
  return (
    <>
      <div className="codingContainer container">
        <h1 className="title">{props.title}</h1>
        <div
          className="codingSection"
          id={props.id}
          //  style={props.poss}
        >
          <div className="codingInfo codingsec">
            <div className="infoCoding infoDetails">{props.info}</div>
            <NavLink to={props.path}>
```

```
            <button className="btn">{props.con}</button>
          </NavLink>
        </div>
        <div className="codingimage codingsec">
          <img src={props.image} alt="image1" className="CodingImg" />
        </div>
      </div>
    </div>
  </> );}
export default CodingPage;
```

(Feedback Page)

```
import React, { useState } from "react";
import blob from "../../assets/blobanimation.svg";
import { toast } from "react-hot-toast";

function Feedback() {
  const [userData, setUserData] = useState({
    name: "",
    email: "",
    feedback: "",
  });

  const [errors, setErrors] = useState({});

  const postUserData = (event) => {
    const { name, value } = event.target;
    setUserData({ ...userData, [name]: value });
  };

  // Validation Function
  const validate = () => {
    const errors = {};

    if (!userData.name.trim()) {
      errors.name = "Name is required";
    }
```

```
  if (!userData.email.trim()) {
    errors.email = "Email is required";
  } else if (!/\S+@\S+\.\S+/.test(userData.email)) {
    errors.email = "Invalid email address";
  }

  if (!userData.feedback.trim()) {
    errors.feedback = "Feedback is required";
  }

  return errors;
};

const submitData = async (e) => {
  e.preventDefault();

  const validationErrors = validate();
  setErrors(validationErrors);

  if (Object.keys(validationErrors).length === 0) {
    try {
      const loadingToast = toast.loading("Submitting...");
      const res = await fetch("http://localhost:5000/api/feedback", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(userData),
      });

      if (res.ok) {
        toast.dismiss(loadingToast);
        toast.success("Feedback Submitted Successfully!");
        setUserData({ name: "", email: "", feedback: "" });
      } else {
```

```
        toast.dismiss(loadingToast);
        toast.error("Server Error! Please try later.");
      }
    } catch (error) {
      console.error(error);
      toast.error("Server Error! Please try later.");
    }
  }
};
return (
  <>
    <div className="container">
      <img src={blob} alt="" className="blob_a blob3" />
      <h1 className="title">Let's Work Together</h1>
      <div className="feedSection">
        <div className="feedinfo">
          <p className="info">
            For Being <mark>Amazing</mark>. Let's Become a part of an amazing
            community <mark>CodeSync</mark>. You can even tell your{" "}
            <mark>Issues Related to Website for Improving the Site</mark>.
          </p>
        </div>
        <div className="feedform">
          <form onSubmit={submitData}>
            <div className="formname formDetails">
              <label htmlFor="name">Name :</label>
              <input
                type="text"
                name="name"
                id="name"
                value={userData.name}
                onChange={postUserData}
              />
              {errors.name && <p style={{ color: "red", fontSize: "14px" }}>{errors.name}</p>}
            </div>
            <div className="formemail formDetails">
```

```
            <label>Email :</label>
            <input
              type="email"
              name="email"
              id="email"
              value={userData.email}
              onChange={postUserData}
            />
            {errors.email && <p style={{ color: "red", fontSize: "14px" }}>{errors.email}</p>}
          </div>
          <div className="formmessage formDetails">
            <label>Message :</label>
            <textarea
              name="feedback"
              id="feedback"
              value={userData.feedback}
              onChange={postUserData}
            />
            {errors.feedback && (
              <p style={{ color: "red", fontSize: "14px" }}>{errors.feedback}</p>
            )}
          </div>
          <div className="formbutton formDetails">
            <button type="submit" className="btn formbtn">
              Say a Word?
            </button>
          </div>
        </form>
      </div>
    </div>
  </div>
  </>
 );
}
export default Feedback;
```

(Header Page)

```
import React, { useContext, useState } from "react";
import { NavLink, useLocation } from "react-router-dom";
import headerImg from "../assets/mainLogo.png";
import profileIcon from "../assets/profileIcon.png";
import { UsedContext } from "./App";

function Header() {
  const { state } = useContext(UsedContext);
  const [isDropdownVisible, setIsDropdownVisible] = useState(false);
  const [isMobileMenuOpen, setIsMobileMenuOpen] = useState(false);

  const toggleDropdown = () => setIsDropdownVisible((prev) => !prev);
  const toggleMobileMenu = () => setIsMobileMenuOpen((prev) => !prev);

  const location = useLocation();
  const isEditorPage = location.pathname.startsWith("/editor");

  const RenderMenu = () => {
    if (state) {
      return (
        <div className="profile-container">
          <img
            src={profileIcon}
            alt="Profile"
            className="profile-icon"
            onClick={toggleDropdown}
          />
          {isDropdownVisible && (
            <div className="profile-dropdown">
              <NavLink to="/profile" className="dropdown-option">
                Profile
              </NavLink>
              <NavLink to="/logout" className="dropdown-option">
                Logout
              </NavLink>
```

```
          </div>
        )}
      </div>
    );
  } else {
    return (
      <>
        <NavLink to="/login">
          <button className="Headerbtn Headerbtn1 btn">Login</button>
        </NavLink>
        <NavLink to="/register">
          <button className="Headerbtn Headerbtn2 btn">Register</button>
        </NavLink>
      </>
    );
  }
};
return (
  <div className="headerContainer">
    <div className="headerImage">
      <img className="headerlogo" src={headerImg} alt="MainLogo" />
      <NavLink className="headerp" to="/">
        CodeSync
      </NavLink>
    </div>

    {/* Only show nav links & hamburger if not on editor page */}
    {!isEditorPage && (
      <>
        {/* Hamburger icon */}
        <div className="hamburger" onClick={toggleMobileMenu}>
          ☰
        </div>

        {/* Desktop Navigation */}
        <div className="languageLinks">
```

```
    <a href="#python">Python</a>
    <a href="#javascript">JavaScript</a>
    <a href="#java">Java</a>
    <a href="#c">C</a>
    <a href="#dart">Dart</a>
    <a href="#image2text">Image2Text</a>
    <a href="#voice2text">Voice2Text</a>
  </div>

  {/* Mobile Dropdown Menu */}
  {isMobileMenuOpen && (
    <div className="mobileMenu">
      <a href="#python" onClick={toggleMobileMenu}>
        Python
      </a>
      <a href="#javascript" onClick={toggleMobileMenu}>
        JavaScript
      </a>
      <a href="#java" onClick={toggleMobileMenu}>
        Java
      </a>
      <a href="#c" onClick={toggleMobileMenu}>
        C
      </a>
      <a href="#dart" onClick={toggleMobileMenu}>
        Dart
      </a>
      <a href="#image2text" onClick={toggleMobileMenu}>
        Image2Text
      </a>
      <a href="#voice2text" onClick={toggleMobileMenu}>
        Voice2Text
      </a>
    </div>
  )}
</>
```

```
      )}

      <div className="Headerbtngroup">
        <RenderMenu />
      </div>
    </div>
  );
}

export default Header;
```

(Footer Page)

```
import React from "react";

function Footer() {
  const date = new Date();
  const year = date.getFullYear();
  return (
    <>
      <div class="wave-container wave">
        <p>
          © {year}, Developed & Designed with ♥ by{" "}
          <a
            className="footer_chinmayee"
            target="_chinmayee"
            href="https://chinmayeemohanty.netlify.app/"
          >
            Chinmayee Mohanty
          </a>
        </p>
      </div>
    </>
  );
}
```

```
export default Footer;
```

(Landing Page)

```
import React from "react";
import blob from "../../assets/blobanimation.svg";

function LandingPage() {
  return (
    <div className="landingContainer container">
      <img src={blob} alt="" className="blob_a blob1" />
      <img src={blob} alt="" className="blob_a blob2" />
      <div className="landingInfo">
        <h1>CodeSync</h1>
        <p>
          <p>
            <em>Think smart. Speak clear. Code strong. Repeat endlessly.</em>
          </p>
        </p>
      </div>
    </div>
  );}export default LandingPage;
```

Backend: -

(Authcontroller.js page)

```
const User = require("../model/userSchema");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const crypto = require("crypto");
const nodemailer = require("nodemailer");
const sendRegistrationEmail = require("../utils/sendRegistrationEmail");

const registerUser = async (req, res) => {
  const { userName, userEmail, password, cpassword, role } = req.body;
  if (!userName || !userEmail || !password || (!cpassword && role)) {
```

```
    return res
      .status(422)
      .send({ Error: "Enter Completed Details for Processing" });
  }

  try {
    const userExist = await User.findOne({ userEmail, status: 1 });
    if (userExist) {
      return res.status(421).json({ Error: "Email already exist" });
    } else if (password !== cpassword) {
      return res.status(420).json({ Error: "Password are not matching" });
    }

    const user = new User({ userName, userEmail, password, role });
    await user.save();
    await sendRegistrationEmail(userEmail, userName);
    res.status(200).json({ success: "User Register Successfully" });
  } catch (err) {
    console.log(`Register Error : ${err}`);
  }
};

const loginUser = async (req, res) => {
  try {
    const { userEmail, password } = req.body;

    if (!userEmail || !password) {
      return res.status(403).json({ error: "Please fill the data" });
    }

    const user = await User.findOne({ userEmail, status: 1 });
    if (!user) {
      return res.status(400).json({ error: "Invalid userName or password!" });
    }

    const isMatch = await bcrypt.compare(password, user.password);
```

```
    if (!isMatch) {
      return res.status(402).json({ error: "Password is incorrect" });
    }

    const token = jwt.sign(
      { _id: user._id, userEmail: user.userEmail },
      process.env.JWT_SECRET,
      { expiresIn: "15d" }
    );
    user.tokens = token;
    await user.save();
    res.status(200).json({
      message: "User login successful",
      token,
      user: { _id: user._id, userName: user.userName },
    });
  } catch (err) {
    console.error(`Login Error: ${err}`);
    res.status(500).json({ error: "Internal Server Error" });
  }
};

const logoutUser = (req, res) => {
  res.status(200).send("User Logged out successfully");
};

const forgotPassword = async (req, res) => {
  const { userEmail } = req.body;

  if (!userEmail) return res.status(400).json({ error: "Email is required" });

  const user = await User.findOne({ userEmail, status: 1 });
  if (!user) return res.status(404).json({ error: "User not found" });

  const resetToken = crypto.randomBytes(32).toString("hex");
  user.resetToken = resetToken;
```

```
  user.tokenExpiry = Date.now() + 3600000;
  console.log("Before save", user.resetToken, user.tokenExpiry);
  await user.save();
  console.log("After save", user);

  const transporter = nodemailer.createTransport({
    service: "gmail",
    auth: {
      user: process.env.EMAIL_USER,
      pass: process.env.EMAIL_PASS,
    },
  });
module.exports = {
  registerUser,
  loginUser,
  logoutUser,
  ForgotPassword,
  resetPassword,
};
```

Database: -

(Conn.js page)

```
const mongoose = require('mongoose');
const DB = process.env.DB;

mongoose.connect(DB)
.then(()=>{
    console.log(`Database Connected Successfully!`);
})
.catch((err)=>{
    console.log(`Chinmayee an error occured .The Error is :${err.message}`);
})
```

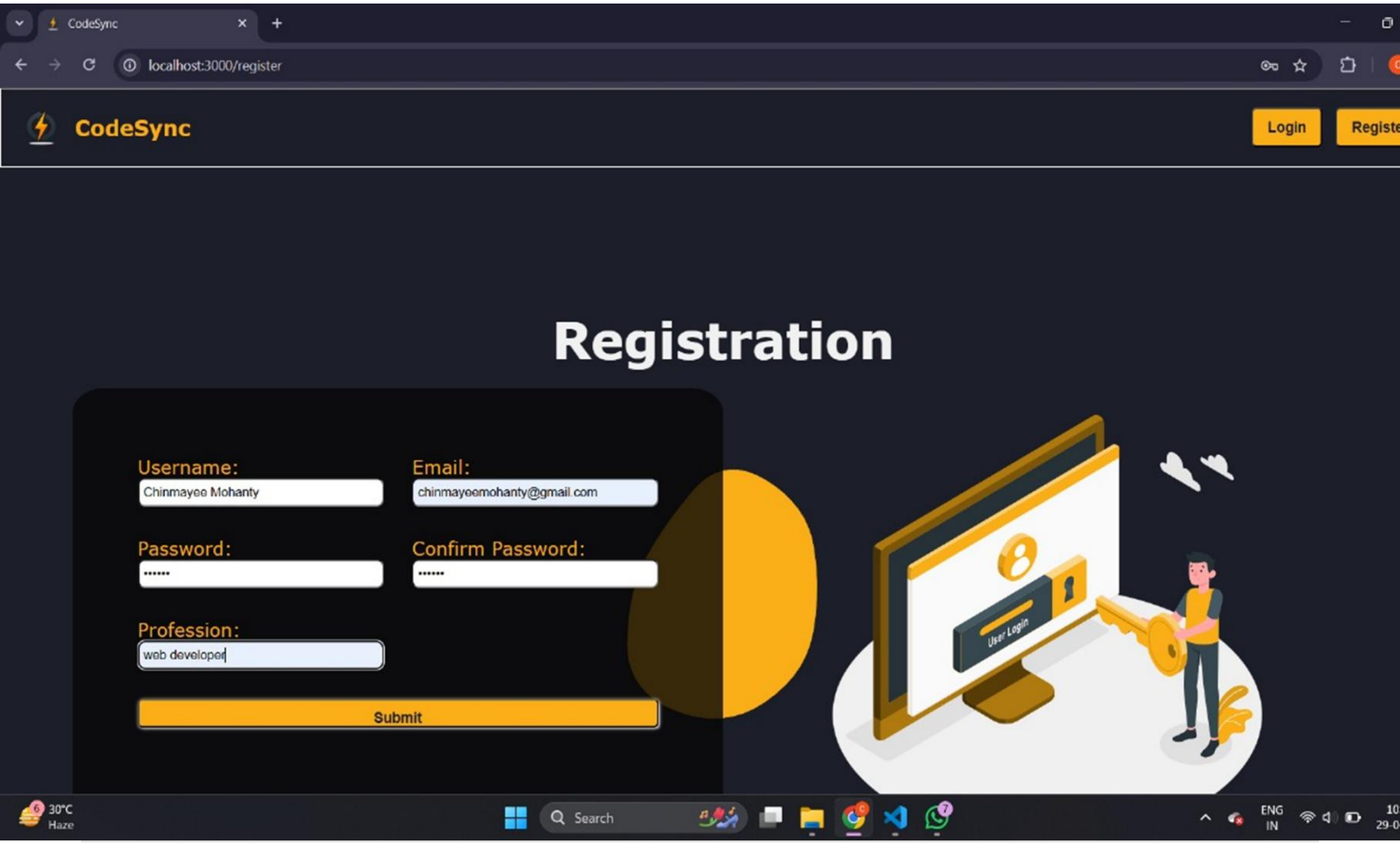
(Feedback Schema .js page)

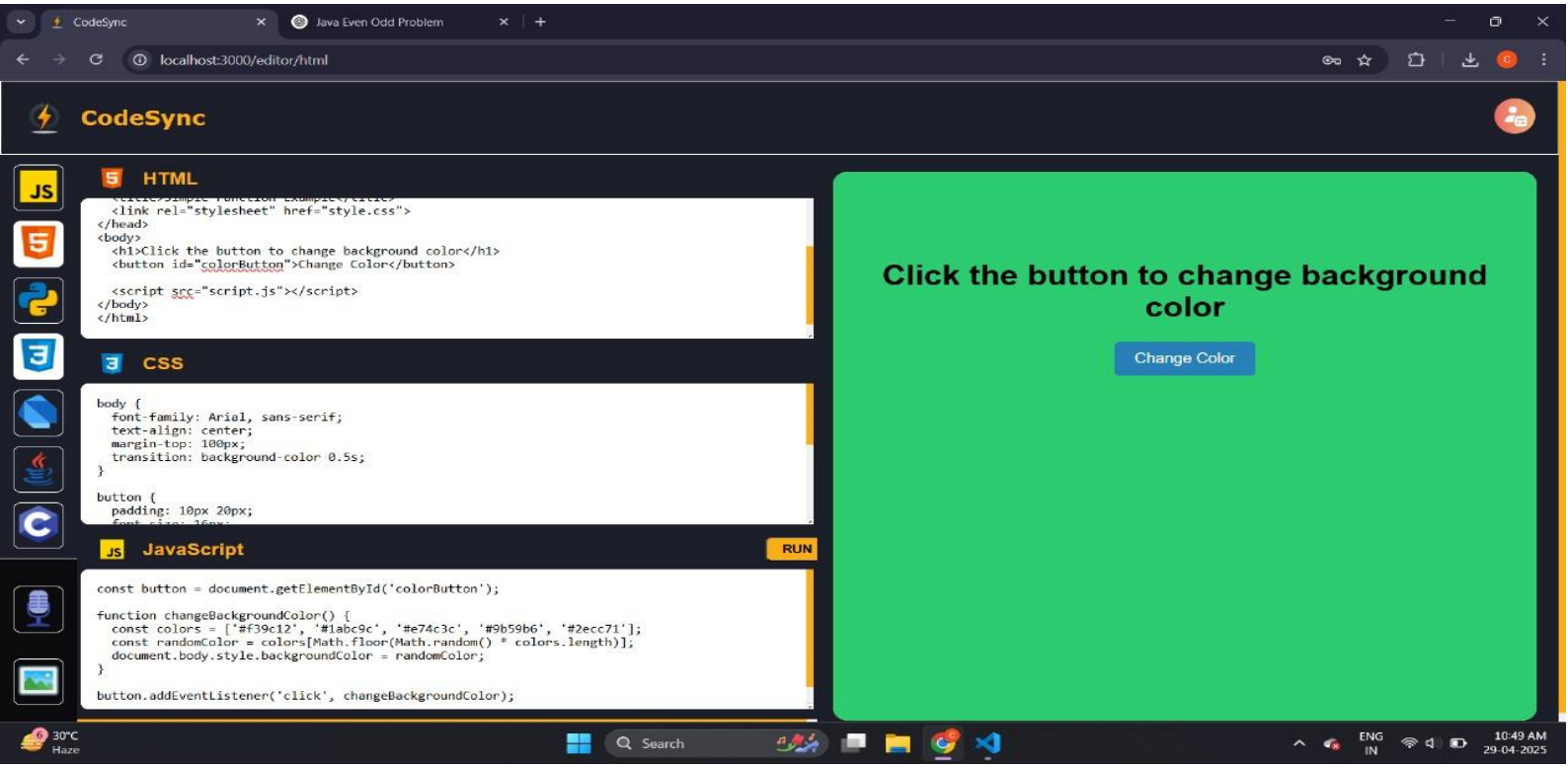
```
const mongoose = require('mongoose');
```

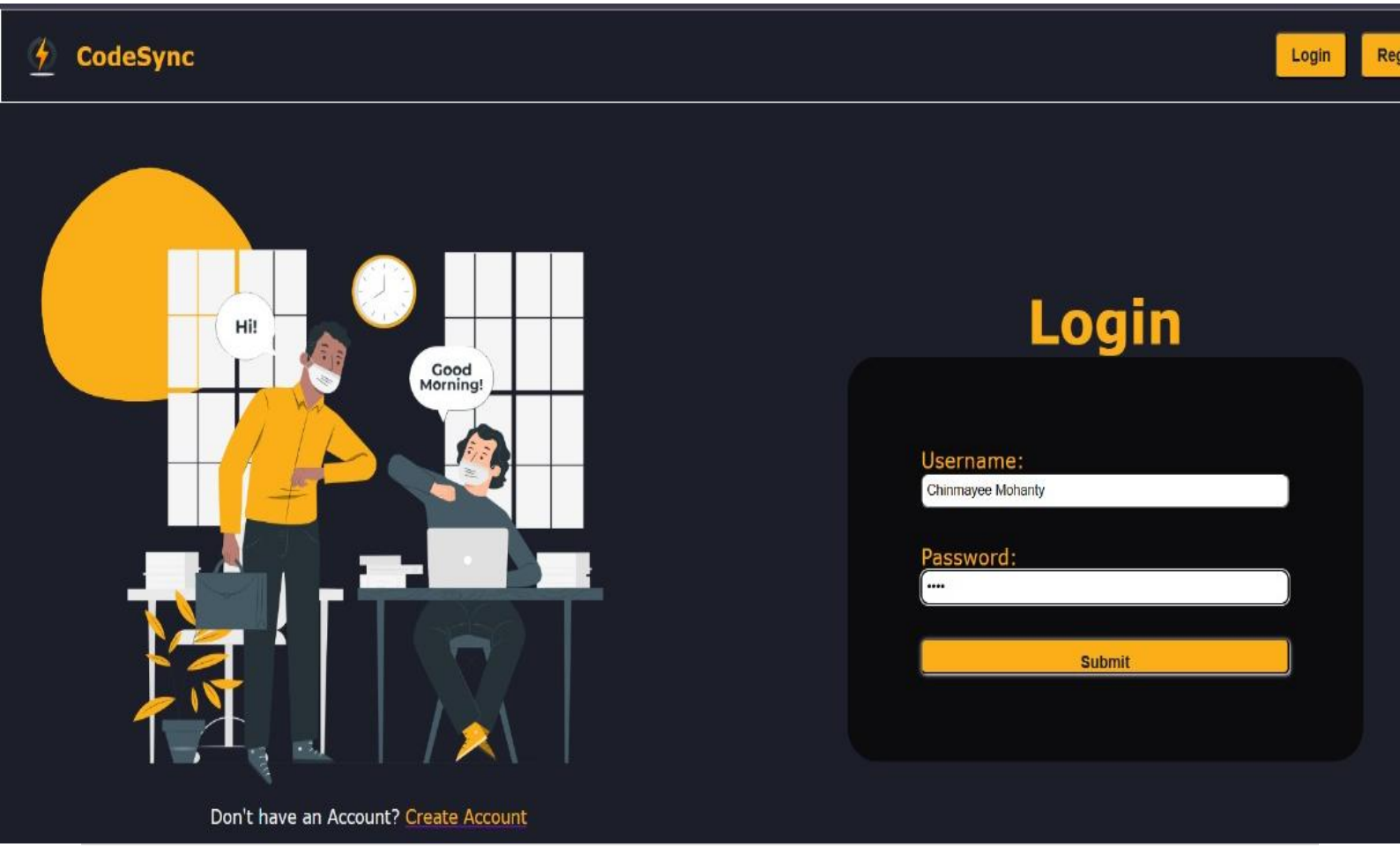
```
const feedbackSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
 },
}, { timestamps: true });

mongoose.connect(DB)
.then(()=>{
    console.log(`Database Connected Successfully!`);
})
.catch((err)=>{
    console.log(`Chinmayee an error occured .The Error is :${err.message}`);
})


module.exports = mongoose.model('Feedback', feedbackSchema);
```

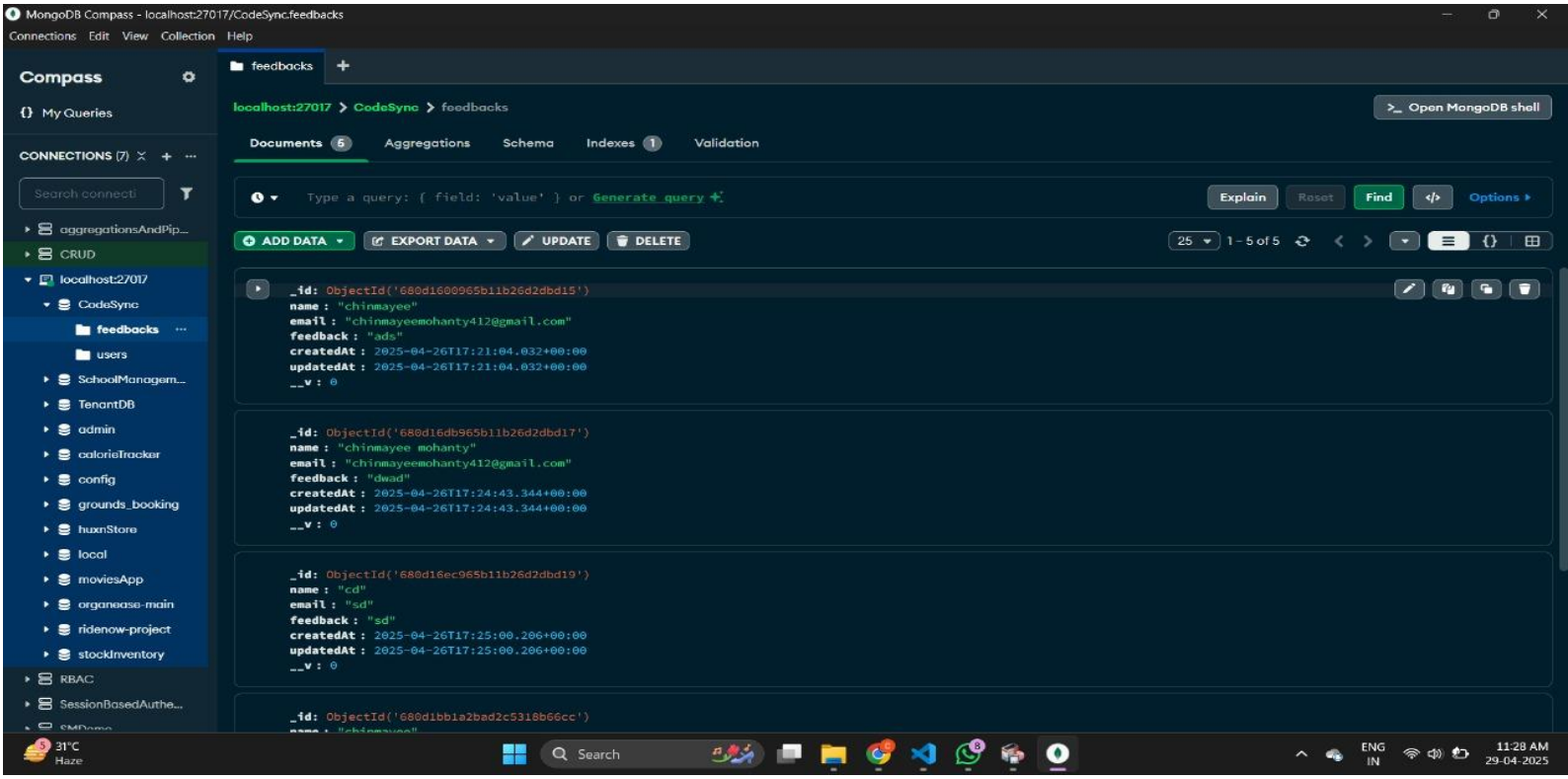
CHAPTER 8

SCREENSHOT OF THE PROJECT




CodeSync
Java Even Odd Problem
localhost:3000/editor/html
CodeSync
HTML
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Click the button to change background color</h1>
<button id="colorButton">Change Color</button>
<script src="script.js"></script>
</body>
</html>
CSS
body {
font-family: Arial, sans-serif;
text-align: center;
margin-top: 100px;
transition: background-color 0.5s;
}
button {
padding: 10px 20px;
JavaScript
RUN
const button = document.getElementById('colorButton');
function changeBackgroundColor() {
const colors = ['#f39c12', '#1abc9c', '#e74c3c', '#9b59b6', '#2ecc71'];
const randomColor = colors[Math.floor(Math.random() * colors.length)];
document.body.style.backgroundColor = randomColor;
}
button.addEventListener('click', changeBackgroundColor);
Click the button to change background color
Change Color
30°C
Haze
Search
ENG
IN
10:49 AM
29-04-2025


CodeSync
Login
Hi!
Good Morning!
Login
Username:
Chinmayee Mohanty
Password:
Submit
Don't have an Account? Create Account


MongoDB Compass - localhost:27017/CodeSync.feedbacks
Connections Edit View Collection Help
Compass
My Queries
CONNECTIONS (7)
Search connecti
aggregationsAndPip...
CRUD
localhost:27017
CodeSync
feedbacks
users
SchoolManagem...
TenantDB
admin
calorieTracker
config
grounds_booking
huxnStore
local
moviesApp
organease-main
ridenow-project
stockInventory
RBAC
SessionBasedAuthe...
feedbacks
localhost:27017 > CodeSync > feedbacks
Open MongoDB shell
Documents 5
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
25
1 - 5 of 5
_id: ObjectId('680d1600965b11b26d2dbd15')
name : "chinmayee"
email : "chinmayeemohanty412@gmail.com"
feedback : "ads"
createdAt : 2025-04-26T17:21:04.032+00:00
updatedAt : 2025-04-26T17:21:04.032+00:00
__v : 0
_id: ObjectId('680d16db965b11b26d2dbd17')
name : "chinmayee mohanty"
email : "chinmayeemohanty412@gmail.com"
feedback : "dwad"
createdAt : 2025-04-26T17:24:43.344+00:00
updatedAt : 2025-04-26T17:24:43.344+00:00
__v : 0
_id: ObjectId('680d16ec965b11b26d2dbd19')
name : "cd"
email : "sd"
feedback : "sd"
createdAt : 2025-04-26T17:25:00.206+00:00
updatedAt : 2025-04-26T17:25:00.206+00:00
__v : 0
_id: ObjectId('680d1bb1a2bad2c5318b66cc')
31°C
Haze
Search
ENG
IN
11:28 AM
29-04-2025

CHAPTER 9

Conclusion and Future Scope

Conclusion

Cody Sync is a helpful and easy-to-use web app that lets people upload, manage, and share code files. It is designed to work well even on devices with low power, such as older computers or mobile phones.

This makes it useful for students, developers, and teams who need a fast and reliable way to share code without needing strong hardware.

During the project, the team followed a clear plan that included steps like planning, design, building the backend, creating the user interface, testing, and launching. A tool called a **Gantt chart** was used to keep track of each task and its timeline. This helped the team stay on schedule and work together smoothly.

Cody Sync uses tools like **Firebase** for storing files, managing user accounts, and saving data. Features as **chunked file uploads** (uploading large files in small parts), and **sharing with permissions** (controlling who can see or edit files) were added to make the app both fast and secure. These features are especially important for people using slower internet or older devices.

The design of the app is simple and clean. The layout is easy to understand, and it works well on both big and small screens. Users can quickly upload a file, view it, and share it with others through a link. This makes the app useful and timesaving.

In short, Cody Sync met its main goal of creating a light and fast app for file sharing. It also created a strong base for adding more features in the future. The project shows that it is possible to build powerful web apps that are still simple, efficient, and user-friendly.

Cody Sync was carefully built to handle **large files** without slowing down. It does this by uploading files in parts (called **chunks**) and loading files only when needed.

The app also allows **secure sharing**, where users can control who sees or downloads their files. All of this is done using cloud services, so the app works fast and does not require a lot of device memory.

The **design** of the app focuses on being clean and easy to use. Even someone with no technical background can use Cody Sync to upload a file and share it through a link. The app works well on all devices—phones, tablets, or computers—making it very flexible and useful in many situations.

In conclusion, Cody Sync reached its goal of being a fast, simple, and reliable tool for sharing code. It works well across all devices, provides secure file handling, and gives a smooth user experience.

The project also sets a strong base for future updates, like adding real-time editing, mobile apps, and more security features. Cody Sync proves that a modern web app can be powerful while still being simple and accessible to everyone.

Future Scope

Cody Sync has a strong base as a simple and effective file sharing tool for code. However, there are many ways it can grow and improve in the future. With technology always changing, there are exciting opportunities to add more features, make it smarter, and serve more users. Below are some key ideas and plans for how Cody Sync can be improved in the future:

1. Real-Time Collaboration

In the future, Cody Sync can allow multiple users to **work on the same file at the same time**, just like in Google Docs. This means people can write, edit, and view code together without needing to download the file. This would be very useful for team projects, coding classes, and remote developers.

2. Version Control Integration

Cody Sync can be connected to platforms like **GitHub or GitLab**. This would let users **track changes, view history, and roll back to earlier versions** of their files. It would make the platform more powerful for developers who need version control features in their workflow.

3. Mobile App Support

Right now, Cody Sync works in a web browser, but a **dedicated mobile app** could give users a better experience on smartphones and tablets. A mobile app can also offer **offline support**, push notifications, and faster performance, making Cody Sync more flexible and accessible.

4. AI Integration

Cody Sync can include **AI-based features**, like automatic code suggestions, bug detection, or smart formatting. This would help users write better code and find errors faster.

AI could also summarize code or recommend improvements, making the tool even more helpful.

5. Advanced Security Features

As Cody Sync grows, keeping files safe becomes even more important. Future updates could add features like **two-factor authentication**, **file expiration dates**, and **end-to-end encryption**.

These features would protect users' data and make the app more trustworthy.

6. Analytics Dashboard

Adding an **analytics dashboard** could give users helpful insights, such as:

- How many times a file was viewed or downloaded
- When and where files are accessed
- Which files are shared the most

This information could help users better manage their work and track activity.

7. Support for More File Types

Currently, Cody Sync focuses on code files. In the future, it could support other file types like PDFs, images, and videos. This would make the app more versatile for all kinds of users—not just developers.

8. Custom User Roles

Cody Sync could allow users to **create teams and assign roles** like admin, editor, or viewer. This would help larger groups organize their work better and control who can do what with each file.

CHAPTER 10

REFERENCES

1. Attanas, D.B. and Monica, H.G. (2012). Effects of greenhouse gases, In *Proc. IOOC-ECOC*, pp. 557-998.

2. Gurudeep, P.R. and Mahin, P. (2009). Risk sensitive estimation model II. *IEEE Transactions on Automatic Control*, 43 (15): 355 - 363.

3. Prakas, K. (2011). Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses. *IEEE Transactions on Automatic Control*, 26(2): 301–320

4. Ram, R., Krishna, S. and Peter, K. (2005a). Risk sensitive estimation and a differential game. *IEEE Transactions on Automatic Control*, 39(9): 1914– 1918.

5. Ram, R., Krishna, S and Peter, K. (2005b). Differential rectification usingcontrol points. *IEEE Transactions on Geoscience and Remote sensing*, 55: 914 – 918.

6. Singh, K. and Robin, R. (2008). A linear- quadratic game approach to estimationand smoothing. In *American Control Conference,* New York. June 20 – 25, 2008, pp. 2818–2822.