

SEVA SANGH

**A PROJECT REPORT
for
Major Project (KCA451)
Session (2024-25)**

Submitted by

**KULKAMAL SINGH
(2300290140081)
KHUSHI
(2300290140088)
HARSH YADAV
(2300290140071)**

**Submitted in partial fulfillment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATION

**Under the Supervision of
Dr. NEELAM RAWAT
Associate Professor**



**Submitted to
DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(APRIL 2025)**

DECLARATION

We hereby declare that the work presented in this report entitled “**Seva Sangh**”, was carried out by us. We have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

We have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. We have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

We affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

KULKAMAL SINGH (2300290140081)

KHUSHI (2300290140088)

HARSH YADAV (2300290140071)

CERTIFICATE

Certified that **Khushi (2300290140088), Kulkamal Singh (2300290140081), Harsh Yadav (2300290140071)** have carried out the project work having “**Seva Sangh**” (**Project-KCA451**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Kulkamal Singh (2300290140081)

Khushi (2300290140088)

Harsh Yadav (2300290140071)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Dr. Neelam Rawat
Associate Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Akash Rajak
Dean
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Seva Sangh ABSTRACT

Seva Sangh is an innovative web application designed to combat poverty by bridging the gap between individuals in need and compassionate donors or volunteers. The platform fosters direct connections, eliminating intermediaries to ensure timely and effective assistance. Equipped with features such as secure donation gateways, volunteer management tools, and impact-tracking dashboards, Seva Sangh promotes transparency and accountability.

The project aims to empower communities by addressing both immediate needs and long-term development goals, offering resources for education, skill-building, and self-reliance. By leveraging technology to create a culture of giving and collaboration, Seva Sangh aspires to be a catalyst for sustainable social change. Through partnerships with NGOs, corporate entities, and local organizations, the platform envisions a scalable model adaptable to diverse communities.

With a user-centric design and a commitment to transparency, Seva Sangh seeks to inspire and mobilize individuals and organizations to work collectively towards a more equitable and poverty-free society.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr. Neelam Rawat** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Akash Rajak**, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Kulkamal Singh (2300290140081)

Khushi (2300290140088)

Harsh Yadav (2300290140071)

TABLE OF CONTENT

1. INTRODUCTION

1.1	Purpose_____	9
1.2	Scope_____	10
1.3	Overview_____	10
1.4	Goals of the Proposed System_____	11
1.5	Background_____	12
1.6	Project Requirements_____	12
1.7	Technology Used_____	13

2. FEASIBILITY STUDY

2.1	Economical Feasibility_____	21
2.2	Technical Feasibility_____	22
2.3	Operational Feasibility_____	23
2.4	Schedule Feasibility_____	24

3. METHODOLOGY

3.1	Sprint-Based Development_____	25
3.2	Iterative and Incremental Development_____	25
3.3	Collaboration and Teamwork_____	25
3.4	Continuous Feedback and Refinement_____	26
3.5	Focus on Flexibility_____	26

4. PROJECT SCREENSHOT AND CODING

4.1	Home Page_____	36
4.2	Sign Up Page_____	37
4.3	Login Page_____	37
4.4	Current Stock_____	38
4.5	Our Work_____	38
4.6	Coding_____	40

5. SYSTEM DESIGN

5.1	Architecture_____	78
5.2	Database Structure_____	80
5.3	Component Interactions_____	81
5.4	Control flow Graph_____	83

6. DEPLOYMENT AND TESTING

6.1	Deployment on Vercel _____	84
6.2	Level of Testing_____	85
6.3	Testing with Postman_____	86

7. INTENDED OUTCOMES

7.1	Functional Deliverables_____	87
7.2	Social Impact_____	87

7.3	Scalability and Sustainability_____	87
7.4	Educational and Professional Growth_____	88
7.5	Positive Recognition_____	88
7.6	Validation and Testing Outcomes_____	88
7.7	Documentation and Learning_____	88

8. CHALLENGES AND RISKS

8.1	Technical Challenge_____	89
8.2	User Adoption Challenges_____	89
8.3	Data Management and Security <u>Risks</u> _____	90
8.4	Project Management Challenges_____	90
8.5	Social and Ethical Risks_____	90
8.6	External Risks_____	90
8.7	Risk Mitigation Strategies_____	91

9. CONCLUSION

9.1	Technical Achievements_____	92
9.2	Social Impact_____	93
9.3	Challenges Addressed_____	93
9.4	Learning Outcomes_____	93
9.5	Future Scope_____	94
9.6	Final Reflection_____	94

10. REFERENCES

10.1	References_____	95
------	-----------------	----

LIST OF FIGURES

1.	Home Page_____	36
2.	Sign Up Page_____	37
3.	Login Page_____	37
4.	Current Stock_____	38
5.	Our Work_____	38
6.	Architectural Design_____	79
7.	Component Interaction_____	82
8.	Control Flow Graph_____	83

CHAPTER 1

INTRODUCTION

1.1 Purpose

Seva Sangh is creating a web-based application to tackle the problem of poverty. The application will connect needy people with donors and volunteers. Aid distribution generally takes a long, complex route through various middlemen and red tape and is often opaque. Seva Sangh is trying to direct talk to avoid middlemen between those who need help and those who want to help.

The platform will address needs in a particular time frame. It will also have a long-term empowering effect. Also, through this platform, a culture of mutual help will be created in various communities. The application ensures resource transparency and optimization by implementing advanced technologies, including a secure payment system and a live tracking dashboard. Moreover, organizations and humans can participate in community service where it will help you to donate back to the community through Seva Sangh.

Seva Sangh's platform will feature personalized user profiles to better match donors, volunteers, and recipients based on specific needs. A built-in feedback and verification system will ensure trust and accountability. AI-powered suggestions will help users find relevant opportunities quickly, while real-time analytics will track the impact of donations and services.

1.2 Scope

The Seva Sangh is a much bigger organization with many wings to serve many types of users. There are beneficiaries, donors, volunteers, etc. The platform will essentially work as a one-stop solution for all poverty alleviation initiatives by.

- Allowing individuals in need to register and list their specific requirements, such as financial aid, educational materials, or basic necessities.
- Providing a platform for donors to contribute either monetarily or in-kind, with options to specify the type of assistance they wish to offer.
- Creating opportunities for volunteers to participate in community-building initiatives, such as mentoring or skill-training programs.
- In the early stages, the platform will focus on particular local communities to check its working. Seva Sangh will grow in other areas based on the experience, and performance and keeping in mind their issues will respond accordingly. The site is built to scale both locally and functionally. In the future, the multilingual version, including local non-profits, will assist in this relevance.

1.3 Overview

Seva Sangh is a platform that respects user privacy while being efficient, transparent, and inclusive. It makes asking for help and giving help easy through a user-friendly interface that connects you to opportunities that matter. App backend handles the user login, track donation, and manage volunteers among other business logics.

Donors can view the impact of their donation through a complete dashboard and beneficiaries can update their profiles and share their updates. People can see and sign up for things based on their skills and availability. Also, the platform has a section for knowledge sharing, which contains educational materials and inspirational stories to encourage bonding.

1.4 Goals of the Proposed System

The primary goals of Seva Sangh are multifaceted and aim to address both short-term and long-term needs of its users:

1. **Efficient Resource Allocation:** Streamline the process of connecting resources with individuals in need, ensuring timely assistance.
2. **Transparency and Trust:** Build confidence among users by providing detailed tracking of donations and their utilization.
3. **Community Empowerment:** Encourage self-reliance by facilitating skill-building opportunities and providing access to educational materials.
4. **Scalable Impact:** Create a replicable model that can be adapted to different regions and challenges, enabling the platform to grow and assist more communities over time.

By meeting these goals, Seva Sangh aspires to become a catalyst for positive societal change, enabling individuals and organizations to collaborate effectively in the fight against poverty.

1.5 Background

Poverty is a complex and multifaceted issue affecting millions worldwide. Despite the efforts of governments, non-profits, and international organizations, a significant portion of aid often fails to reach the intended recipients due to inefficiencies, corruption, and logistical challenges. This problem is further exacerbated by a lack of transparency in the allocation and utilization of resources.

Seva Sangh was conceptualized to address these gaps by leveraging modern technology to create a streamlined, transparent, and impactful system. Inspired by the success of crowdfunding platforms and community-driven initiatives, Seva Sangh aims to combine the best practices of these models into a holistic solution for poverty alleviation. By focusing on direct connections and eliminating intermediaries, the platform ensures that aid reaches the right individuals without unnecessary delays or complications.

1.6 Project Requirements

The requirements for Seva Sangh are divided into functional, non-functional, and technical categories:

- **Functional Requirements:**
 - i. User registration and authentication for beneficiaries, donors, and volunteers.
 - ii. A secure payment gateway for monetary donations.
 - iii. Features to manage and track donations and volunteer activities.

- iv. Notifications and updates for users about their contributions and impacts.

- **Non-Functional Requirements:**

- i. High availability and reliability to handle user requests at any time.
- ii. Scalability to accommodate increasing users and functionalities as the platform grows.
- iii. Technical Requirements:
- iv. Use of robust web development frameworks such as Django or React.
- v. Integration with secure third-party APIs for payment processing and data analytics.
- vi. Cloud-based storage solutions for scalability and data security.

1.7 Technology Used

Seva Sangh relies on cutting-edge technologies to ensure the platform is reliable, scalable, and user-friendly:

- **Frontend:** HTML, CSS, React.Js, Next.Js for a responsive and intuitive user interface.
- **Backend:** Python with Django or Flask for efficient server-side processing.
- **Database:** MongoDB for storing user data, donation records, and impact metrics.

- **Security Tools:** SSL encryption, secure payment gateways, and user authentication protocols to safeguard user data.

By combining these technologies, Seva Sangh creates a robust platform capable of addressing the diverse needs of its users and driving impactful social change.

PROJECT REQUIREMENTS

A system is developed after gathering and analysing the user requirements of the system. The first most part of system development is to gathering information about user by doing preliminary investigation which is starting investigation about user requirement. Analysts use different techniques to collect data during primary investigations.

1) Documents Reviewing Organization

The analysts conducting the investigation first learn the organization involved in, or affected by the project. Analysts can get some details by examining organization charts and studying written operating procedures.

Collected data is usually of the current operating procedure:

- The information relating to clients, projects and students and the relationship between them was held manually.
- Managing of follow-ups was through manual forms.
- Complaints require another tedious work to maintain and solve.
- Payments details had to be maintained differently.

2) Gathering Information By Asking Questions

Interviewing is the most commonly used techniques in analysis. It is always necessary first to approach someone and ask them what their problems are, and later to discuss with them the result of your analysis.

3) Questionnaires

Questionnaires provide an alternative to interviews for finding out information about a system. Questionnaires are made up of questions about information sought by analyst. The questionnaire is then sent to the user, and the analyst analyzes the replies.

4) Electronic Data Gathering

Electronic communication systems are increasingly being used to gather information. Thus it is possible to use electronic mail to broadcast a question to a number of users in an organization to obtain their viewpoint on a particular issue.

In my project, with the help of Marg software solutions, I have send questionnaire through electronic mail to twenty employees of the company and retrieved the information regarding the problem faced by existing system.

5) Interviews

Interview allows the analysts to learn more about the nature of the project request and reason of submitting it. Interviews should provide details that further explain the project and show whether assistance is merited economically, operationally or technically.

One of the most important points about interviewing is that what question you need to ask.

It is often convenient to make a distinction between three kinds of question that is

- a. Open questions
- b. Closed question
- c. Probes

Open questions are general question that establish a person's view point on a particular subject.

Closed questions are specific and usually require a specific answer.

Hardware requirements	
Processor	RAM
Core i3,i5 or i7	4 GB or more

Software requirements	
Operating system	Software requirement
Windows 7,8,10, Linux, or any other higher version	Google chrome, internet explorer, or any web browser

TECHNOLOGIES USED

1) HTML, CSS, and JavaScript:

- i. Used for creating the structure, styling, and interactivity of the user interface.
- ii. Ensures a responsive and visually appealing design.

2) React.js:

- i. A popular JavaScript library for building dynamic and fast user interfaces.
- ii. Enables component-based development, making the frontend modular and reusable.

3) MongoDB:

- i. A NoSQL database used for storing and managing data in a flexible, document- oriented format.
- ii. Suitable for handling large-scale, unstructured, or semi-structured data like user profiles, donation records, and volunteer activities.
- iii. Provides features like scalability, high availability, and efficient querying, making it ideal for modern web applications.

4) Git and GitHub:

- i. Used for version control and collaborative development.
- ii. Ensures efficient code management and team synchronization.

5) Postman:

- i. Postman for testing APIs and ensuring reliable backend functionality.

6) NEXTJS:

Next.js is a powerful React-based framework ideal for building fast and scalable web applications like Seva Sangh. By leveraging features like server-side rendering (SSR), static site generation (SSG), and API routes, Next.js helps enhance performance, improve SEO, and simplify backend integrations.

CHAPTER-2

FEASIBILITY STUDY

When the initial investigation points to potential value, the process moves into a more in- depth feasibility study. This study tests the viability of a system proposal by assessing its practicality, impact on the organization, ability to meet user needs, and efficient use of resources. Essentially, it answers critical questions like:

- What specific needs do users have, and how well does the proposed system address them?
- What resources (e.g., time, budget, skills) are available to support the system?
- What impact will the system have on the organization, both positive and negative?
- Is solving this problem worthwhile given the costs and benefits?

Key Steps in the Feasibility Analysis

The process typically involves the following eight steps:

1. Form a Project Team and Assign Leadership
2. Create System Flowcharts
3. List Potential System Proposals
4. Define Characteristics of Proposed Systems
5. Evaluate Performance and Cost-Effectiveness
6. Compare System Performance and Costs
7. Select the Best System
8. Report Findings to Management

2.1 Economical Feasibility

Overview: Economical feasibility examines the financial aspects of the project, including development costs, potential revenue streams, and return on investment.

2.1.1 Key Points:

Development Costs: Estimate the costs associated with development, including software licenses, developer salaries, infrastructure costs, etc. Compare these costs with the available budget to determine financial feasibility.

Revenue Streams: Identify potential revenue streams for the application, such as subscription plans, advertising, affiliate marketing, etc. Evaluate the market demand and competition to assess the viability of these revenue streams.

Scalability and ROI: Consider the scalability of the application in terms of user growth and revenue generation potential. Calculate the projected return on investment (ROI) based on revenue projections and development costs.

2.1.2 Challenges and Solutions:

Cost Estimation: Accurately estimating development costs and projecting revenue streams can be challenging due to uncertainties in market dynamics and user adoption. To address this, a detailed cost-benefit analysis and market research were conducted to inform financial projections.

Monetization Strategies: Identifying viable monetization strategies requires understanding user needs and market trends. Experimentation with different monetization models and continuous refinement based on user feedback and market analysis helped identify effective revenue streams.

2.2 Technical Feasibility

Overview: Technical feasibility assesses whether the proposed system can be successfully developed and implemented using the available technology and resources.

2.2.1 Key Points:

Technology Stack: Evaluate the suitability of the chosen technologies (React.js for frontend, JavaScript for backend) for building the application. Consider factors such as compatibility, scalability, performance, and support.

Development Tools: Assess the availability and proficiency of development tools, libraries, and frameworks required for implementing the features of the application. Consider factors such as IDEs, version control systems, testing frameworks, etc.

Integration with External APIs: Evaluate the feasibility of integrating external APIs for fetching real-time data. Consider factors such as API documentation, rate limits, authentication mechanisms, and data format compatibility.

2.2.2 Challenges and Solutions:

Technology Selection: Choosing appropriate technologies that align with project requirements and development team expertise is crucial. To

address this, thorough research and prototyping were conducted to evaluate different technology options and assess their suitability for the project.

API Integration: Integrating external APIs for real-time data retrieval can pose challenges such as handling rate limits, managing authentication tokens, and processing large volumes of data. To overcome this, rate-limiting strategies, caching mechanisms, and efficient data processing techniques were implemented.

2.3 OPERATIONAL FEASIBILITY

Overview: Operational feasibility assesses whether the proposed system can be effectively integrated into existing operational processes and workflows.

2.3.1 Key Points:

User Adoption: Evaluate the willingness of users to adopt the new system and any potential resistance to change. Consider factors such as user training, usability, and perceived benefits of the new system.

Integration with Existing Systems: Assess the compatibility of the new system with existing infrastructure, software, and workflows. Identify potential challenges and opportunities for integration.

Scalability and Maintenance: Consider the long-term scalability and maintenance requirements of the system. Evaluate the availability of resources and expertise required to support ongoing operations and updates.

2.3.2 Challenges and Solutions:

Change Management: Addressing user resistance to change and ensuring smooth transition to the new system requires effective change management strategies. This involved conducting user training sessions, providing documentation and support resources, and soliciting feedback from stakeholders.

Legacy System Integration: Integrating the new system with existing legacy systems can be complex due to differences in technology stacks, data formats, and business processes. To overcome this, APIs, middleware, and data migration tools were utilized to facilitate seamless integration and data exc

CHAPTER-3

METHODOLOGY

For Seva Sangh, Agile methodology was tailored to suit the context of a college project, emphasizing flexibility, collaboration, and iterative progress without external stakeholders or investors. Here's how it was applied:

3.1 Sprint-Based Development:

- The development process was divided into short, time-boxed sprints (e.g., 1- 2 weeks each), with a clear focus on specific tasks or features.
- Each sprint concluded with a deliverable, such as completing the user interface, integrating MongoDB, or testing a feature.

3.2 Iterative and Incremental Development:

- **Iteration 1:** Focused on creating the foundational architecture, including setting up MongoDB and basic backend functionalities.
- **Iteration 2:** Developed core features like user registration, donation forms, and volunteer opportunities.
- **Iteration 3:** Enhanced the platform with features like impact tracking dashboards and user authentication.
- **Iteration 4:** Conducted final testing and integrated feedback for a polished output.

3.3 Collaboration and Teamwork:

- Team members worked collaboratively, dividing responsibilities like frontend design, backend development, and testing.

- Regular team discussions were held to review progress, identify challenges, and propose solutions.

3.4 Continuous Feedback and Refinement:

- Feedback was sought during each sprint, either through peer reviews or by testing the application against predefined goals.
- Issues identified during testing were added to the backlog and resolved in subsequent sprints.

3.5 Focus on Flexibility:

- Agile's flexibility allowed for adapting to changes in requirements, such as adding a new feature or modifying an existing one based on the project scope.
- Priorities were adjusted dynamically to ensure the project aligned with the college's objectives and timelines.

System Analysis

System is created to solve problems. One can think of the systems approach as an organized way of dealing with a problem. In this dynamic world, the subject system analysis and design, mainly deals with the software development activities.

Since a new system is to be developed, the one most important phases of software development life cycle is system requirement gathering and analysis. Analysis involves detailed study of the current system, leading to specification of a new system. Analysis is a detailed study of various

operations performed by a system and their relationship within and outside the system. Using the following steps it becomes easy to draw the exact boundary of the new system under consideration.

Keeping in view the problems and new requirements, workout the pros and cons including new area of the system.

All procedures, requirements must be analyzed and documented in the form of detailed DFDs, logical data structure and miniature specifications.

System analyses also include sub-dividing of complex process involving the entire system, identification of data store and manual processes.

System Analysis is conducted with the following steps

- Information gathering
- The tools of structured analysis
- Identification of Need
- System Planning and initial investigation
- Feasibility study

1. Information Gathering:

- Information about the firm
- Information about the workflow
- Various tools used are:
 - Review of literature
 - Procedure
 - Forms

Initial investigation:

- Problem definition and project initiation
- Determining the requirements

- Needs identification
- Dimension of planning
- Determination of feasibility

Feasibility Analysis:

- System Performance definition
- Identification of system objectives
- Description of outputs

Preliminary Investigation:

- Evaluation of project request is major purpose of preliminary investigation.
- It is the collecting information that helps committee members to evaluate merits of the project request and make judgment about the feasibility of the proposed projects.
- To answer the above questions, system analysts discuss with different category of person to collect facts about their business and their operations.
- When the request is made, the first activity the preliminary investigation begins.

Preliminary investigation has three parts-

1. Request Clarification: An information system is intended to meet needs of an organization. Thus the first step in this phase is to specify these needs and requirements.
2. The next step is to determine the requirements met by the system. Many requests from employees and users in the organizations are not clearly

defined. Therefore, it become necessary that project request must examine and clarified properly before considering system investigation.

3. Information related to different needs of the System can be obtained by different users of the system. This can be done by reviewing different organization's documents such

SDLC

Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying and maintaining the software.

SDLC defines the complete cycle of development i.e. all the tasks involved in planning, creating, testing, and deploying a Software Product.

SDLC Phases

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

1) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

For Example: A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

2) Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

5) Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

6) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e., if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

SOFTWARE ENGG. PARADIGM APPLIED

Software engineering is a layered technology. The foundation for software engineering is the process layer. Software engineering processes the glue that holds the technology layers together and enables ratios and timely

development of computer software. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology.

Software engineering methods provide the technical how-top's for building software.

Method compass a broad array of tasks that include requirements analysis, design, program construction, testing and support. Software engineering tools provide automated or semi- automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another tool, a system for the support of software development, called computer-aided software engineering is established.

The following paradigms are available:

1. Waterfall Model
2. Spiral model
3. Agile Model

As stated before, for seva sangh, The Agile Method is used-

Agile Methodology

Agile methodology is a modern approach to software development that emphasizes flexibility, collaboration, and iterative progress. Unlike traditional methods like the

Waterfall model, Agile is adaptive, allowing teams to respond to changes and deliver value incrementally. It is widely used in projects where requirements may evolve over time.

Key Principles of Agile Methodology-

Agile is based on the Agile Manifesto, which focuses on the following principles:

1. Individuals and Interactions Over Processes and Tools: Collaboration among team members is prioritized over rigid processes.
2. Working Software Over Comprehensive Documentation: Deliver functional
3. software frequently rather than spending excessive time on documentation.
4. Customer Collaboration Over Contract Negotiation: Close interaction with customers ensures their needs are met.

Responding to Change Over Following a Plan: Flexibility to adapt to changes, even late in the development process.

Core Features of Agile:

1. Iterative Development:
 - The project is divided into smaller, manageable cycles called sprints (typically 1–4 weeks).
 - Each sprint delivers a functional increment of the product.
2. Collaborative Environment:
 - Continuous communication between developers, testers, and customers ensures alignment.
3. Flexibility:
 - Agile embraces change, allowing teams to modify priorities or features as requirements evolve.
4. Customer Involvement:
 - Regular demonstrations and feedback loops ensure the product aligns with user needs.

5. Focus on Quality:

- Continuous testing is integrated into each sprint, ensuring that defects are caught and fixed early.

Agile Methodologies:

Several frameworks and methodologies are based on Agile principles, including:

1. **Scrum:** Focuses on short sprints, daily stand-ups, and a defined team structure with roles like Scrum Master and Product Owner.
2. **Kanban:** Uses a visual board to manage workflow and optimize efficiency.
3. **Extreme Programming (XP):** Emphasizes frequent releases, test-driven development (TDD), and close collaboration.
4. **Lean:** Focuses on minimizing waste and maximizing value.

Steps in Agile Development:

1. Concept and Planning:

- Define high-level goals, create a product backlog, and prioritize user stories.

2. Sprint Planning:

- Select tasks for the sprint and define deliverables.

3. Sprint Execution:

- Develop, test, and integrate features within the sprint cycle.
- Daily stand-up meetings help track progress and resolve roadblocks.

4. Review and Demo:

- Present the sprint's output to stakeholders or team members for feedback.

5. Retrospective:

- Reflect on what went well, what didn't, and identify areas for improvement.

Benefits of Agile:

1. Customer Satisfaction: Continuous delivery of valuable features keeps users engaged.
2. Flexibility: The ability to adapt to changes ensures the product stays relevant.
3. Improved Collaboration: Regular communication fosters teamwork and transparency.
4. Faster Time-to-Market: Incremental releases ensure quicker delivery of usable software.
5. Quality Assurance: Continuous integration and testing enhance product quality.

Challenges in Agile:

1. Team Dependency: Requires high coordination, which may be challenging in distributed teams.
2. Scope Creep: Frequent changes can disrupt timelines if not managed effectively.
3. Demanding Time Commitment: Regular meetings and updates require team members to be consistently involved.

CHAPTER-4

PROJECT SCREENSHOTS & CODING

4.1 HOME PAGE

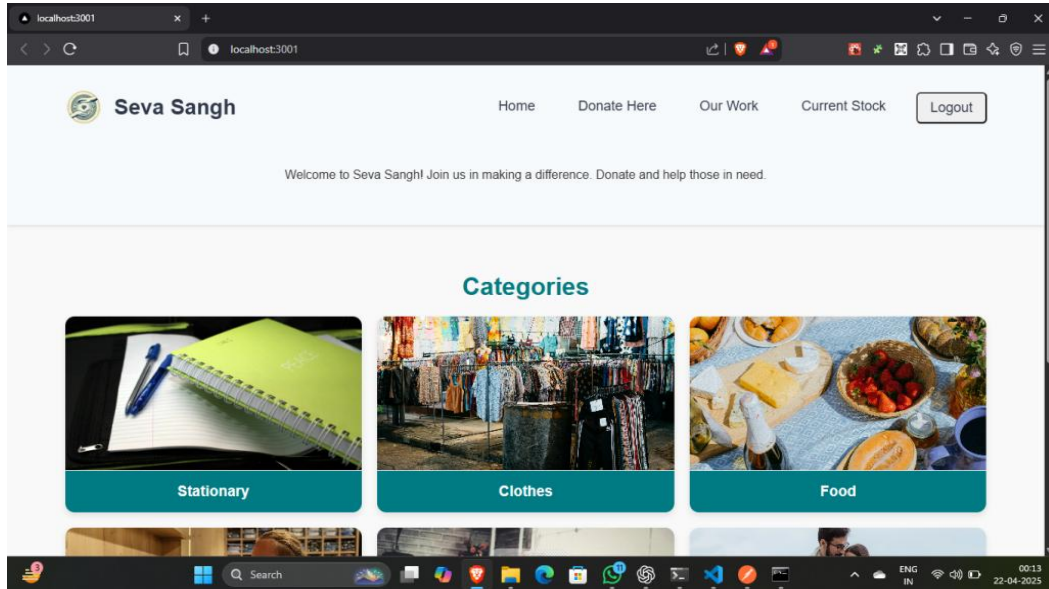


Fig4.1 HOME PAGE

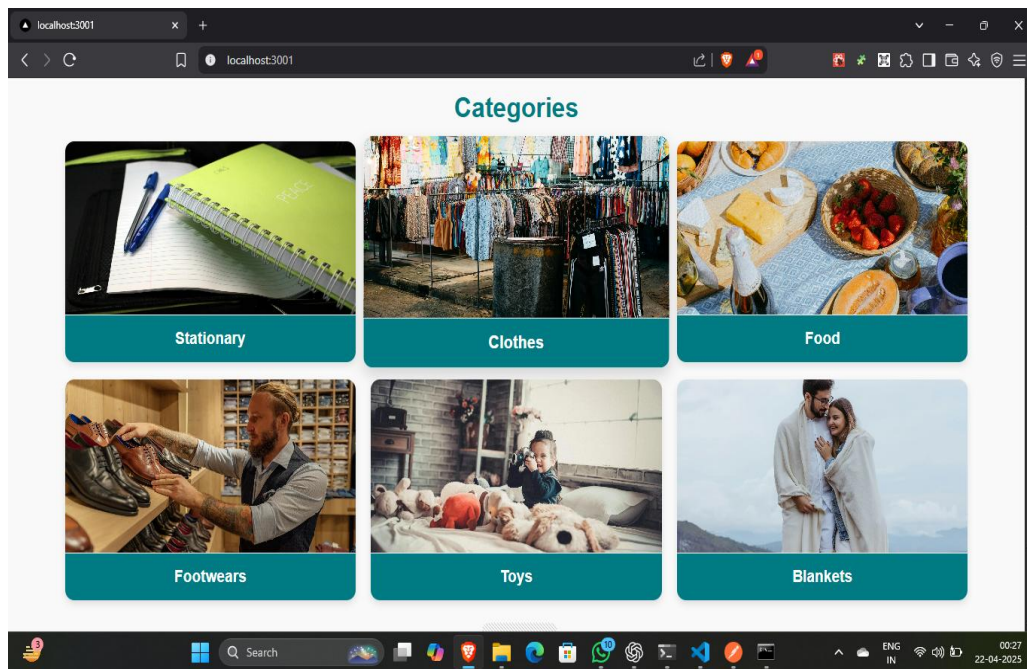
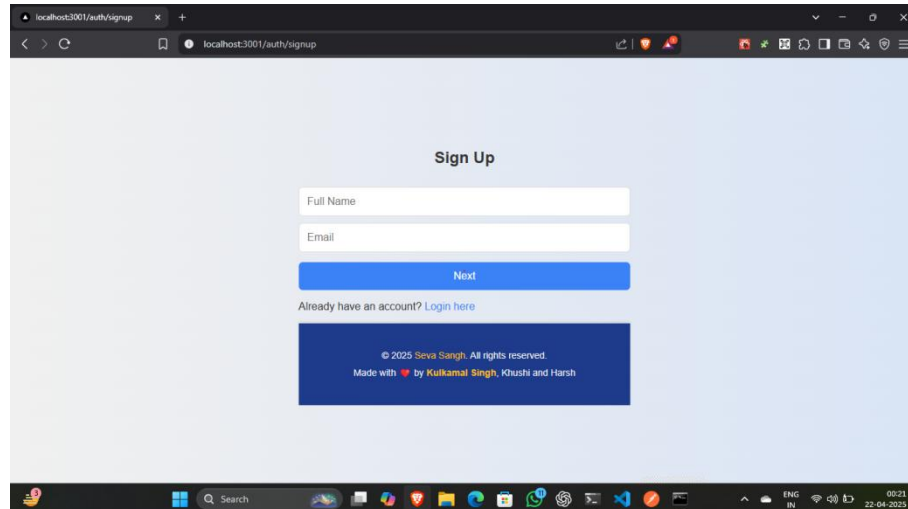


Fig4.1 HOME PAGE

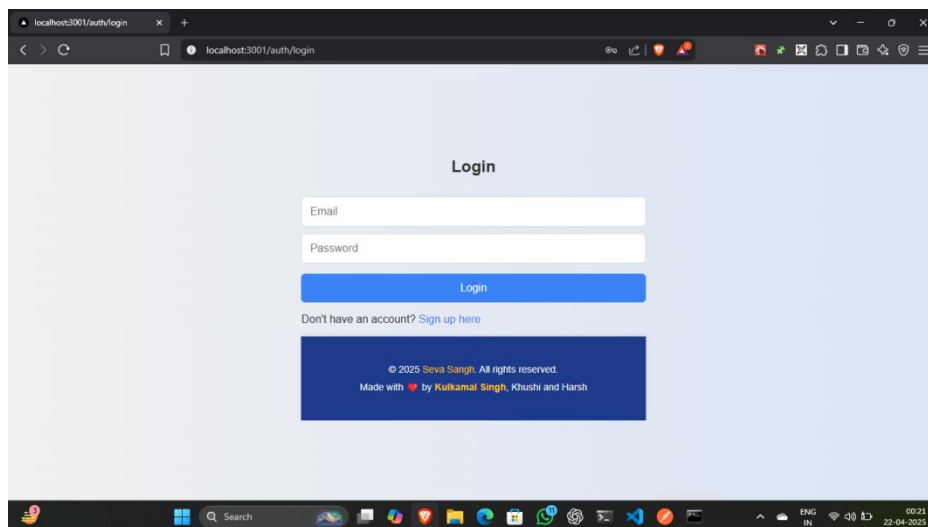
4.2 SIGNUP PAGE



A screenshot of a web browser displaying the 'Sign Up' page. The browser's address bar shows 'localhost:3001/auth/signup'. The page has a light blue background. In the center, there is a white box containing the title 'Sign Up' in bold. Below the title are two input fields: 'Full Name' and 'Email'. A blue button labeled 'Next' is positioned below the 'Email' field. Underneath the button, there is a link that says 'Already have an account? Login here'. At the bottom of the white box, there is a dark blue footer area with white text that reads: '© 2025 Seva Sangh. All rights reserved. Made with ❤️ by Kulkamal Singh, Khushi and Harsh'. The Windows taskbar is visible at the bottom of the screen, showing the time as 00:21 on 22-04-2025.

Fig4.2 SIGNUP PAGE

4.3 LOGIN PAGE



A screenshot of a web browser displaying the 'Login' page. The browser's address bar shows 'localhost:3001/auth/login'. The page has a light blue background. In the center, there is a white box containing the title 'Login' in bold. Below the title are two input fields: 'Email' and 'Password'. A blue button labeled 'Login' is positioned below the 'Password' field. Underneath the button, there is a link that says 'Don't have an account? Sign up here'. At the bottom of the white box, there is a dark blue footer area with white text that reads: '© 2025 Seva Sangh. All rights reserved. Made with ❤️ by Kulkamal Singh, Khushi and Harsh'. The Windows taskbar is visible at the bottom of the screen, showing the time as 00:21 on 22-04-2025.

Fig4.3 LOGIN PAGE

4.4 CURRENT STOCK

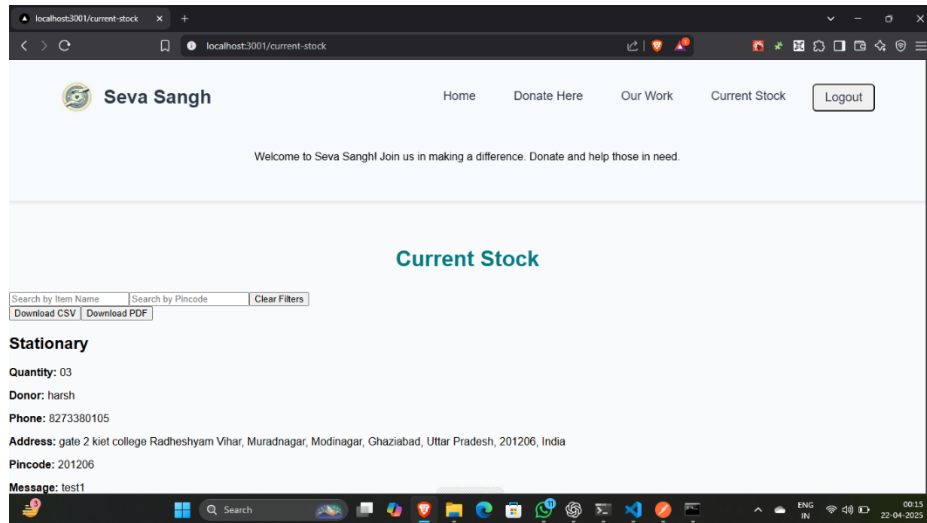


FIG 4.4 CURRENT STOCK

4.5 OUR WORK

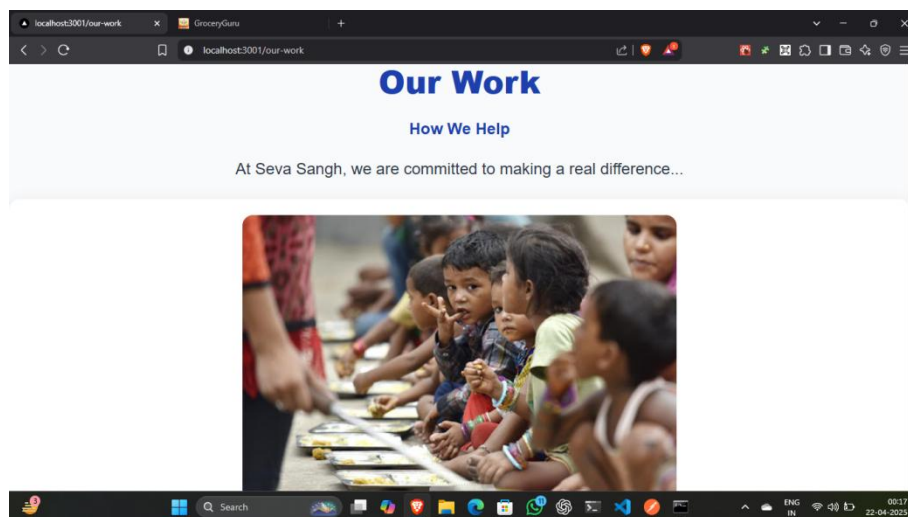


Fig 4.5 OUR WORK



Fig 4.5 OUR WORK

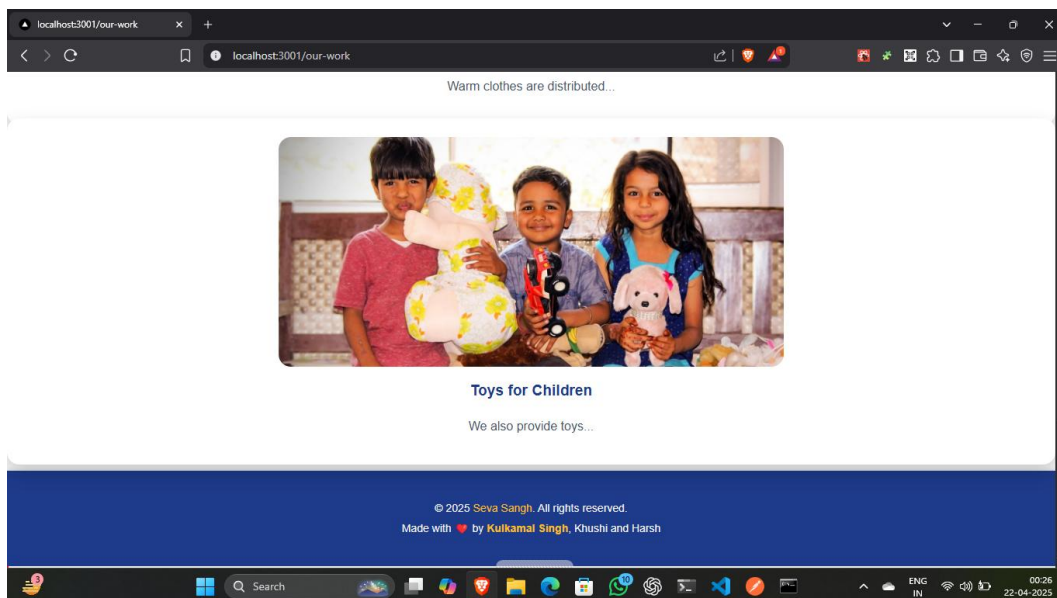


Fig 4.5 OUR WORK

CHAPTER-4

CODING

BACKEND

(codeAuthcontroller.js)

```
import { sendResetPasswordEmail, sendSuccessEmail, sendVerificationEmail,
sendWelcomeEmail } from "../Email/email.js";
import { User } from "../models/userModel.js";
import bcryptjs from 'bcryptjs';
import crypto from 'crypto';

import dotenv from 'dotenv'
import { generateTokenAndSetCookie } from
"../utils/generateTokenAndSetCookie.js";
dotenv.config()
export const signup=async(req,res)=>{
  const {email,password,name} = req.body;
  try {
    if(!email||!name||!password){
      throw new Error("All fields are required")
    }

    const userAlreadyExist=await User.findOne({email})
    if(userAlreadyExist){
      return res.status(400).json({success: false,message: "User already
exists"})
    }
  }
```

```

    }
    const hashedPassword = await bcryptjs.hash(password,10);
    const verificationToken= Math.floor(
        100000 + Math.random() * 900000
    ).toString();

    const user= new User(
        {
            email,
            password: hashedPassword,
            name,
            verificationToken:verificationToken,
            verificationTokenExpiresAt:Date.now() + 24 * 60 * 60 * 1000,
        }
    )
    await user.save();
    await generateTokenAndSetCookie(res,user._id)
    sendVerificationEmail(user.email,verificationToken)
    res.status(201).json({ success:true, message:"User created successfully",
        user:{
            ...user._doc,
            password:undefined
        }
    })

    } catch (error) {
    res.status(400).json({ success: false,message:error.message})

    }

```

```

}
export const forgotPassword=async(req,res)=>{
  const {email}= req.body;
  try {
    const user= await User.findOne({email})

    if(!user){
      return res.status(400).json({ success: false,message:"User not found" })
    }
    const resetToken= crypto.randomBytes(20).toString('hex')
    const resetTokenExpires= Date.now()+1*60*60*1000
    user.resetToken=resetToken
    user.resetTokenExpiration=resetTokenExpires
    await user.save();
    await
    sendResetPasswordEmail(user.email,`${process.env.RESET_URL}/forgot-
password/${resetToken}`)
    res.status(200).json({ success:true, message:"Reset password link sent
successfully" })

  } catch (error) {
    console.log("Error sending mail",error);

    res.status(400).json({ success: false,message:error.message })

  }
}
export const resetPassword=async(req,res)=>{
  const {token}=req.params;
  const {password}=req.body;
  try {

```

```

const user= await User.findOne(
  {
    resetToken:token,
    resetTokenExpiration:{ $gt:Date.now()}
  }
)
if(!user) {
  return res.status(400).json({ success: false,message:"Invalid or Expired
Token"})
}
const hashedPassword = await bcryptjs.hash(password,10);
user.password=hashedPassword;
user.resetToken=undefined;
user.resetTokenExpiration=undefined;
await user.save();
await sendSuccessEmail(user.email)

res.status(200).json({ success:true, message:"Password reset successfully" })

} catch (error) {
  console.log("error resetting password",error);
  res.status(400).json({ success: false,message:"Password reset failed" })

}
}

export const verifyEmail=async(req,res)=>{
  const {code}=req.body;
  try {
    const user= await User.findOne(

```

```

        {
          verificationToken:code,
          verificationTokenExpiresAt:{$gt:Date.now()}
        }
      )
      if(!user){
        return res.status(400).json({ success: false,message:"Invalid or Expired
Token" })
      }
      user.isVerified = true;
      user.verificationToken=undefined;
      user.verificationTokenExpiresAt=undefined;
      await user.save()
      await sendWelcomeEmail(user.name, user.email)

      res.status(200).json({ success:true, message:"Email verified successfully" })
    } catch (error) {
      res.status(400).json({ success:false, message:"Verification failed" })
    }

  }
}

export const checkAuth=async(req,res)=>{
  try {
    const user = await User.findById(req.userId).select("-password");
    if (!user)
      return res
        .status(401)
        .json({ success: false, message: "User not found" });
    res.status(200).json({ success: true, user });
  } catch (error) {

```

```

    console.log("error in checkAuth", error);

    res
      .status(500)
      .json({ success: false, message: "Error checking authentication" });
  }
}

export const login=async(req,res)=>{
  const {email , password}=req.body;
  console.log(req.body);

  if(!email||!password){
    return res.status(400).json({ success: false,message:"All fields are
required" })
  }
  try {
    const user=await User.findOne({email})
    if(!user){
      console.log("user not found");

      return res.status(400).json({ success: false,message:"User not found" })
    }
    if(!user.isVerified){
      return res.status(400).json({ success: false,message:"Email is not
Verified" })
    }

    const isPasswordValid= await bcryptjs.compare(password,user.password);
    if(!isPasswordValid){

```

```

    console.log("not valid");

    return res.status(400).json({ success: false,message:"Invalid credentials"})
  }
  console.log("pass check successful");

  await generateTokenAndSetCookie(res,user._id)
  user.lastLogin = new Date()
  await user.save();
  res.status(200).json({ success:true, message:"Logged in successfully",
    user:{
      ...user._doc,
      password:undefined
    }
  })
}
catch (error) {
  console.log("Login Failed",error);

  res.status(400).json({ success: false,message:error.message})

}

}

export const logout=async(req,res)=>{
  res.clearCookie('authToken')
  res.json({ success:true, message:"Logged out successfully"})
}

```

(Communitycontroller.js)

```
import Community from "../models/communityModel.js";
import { User } from "../models/userModel.js";
import io from "../index.js"

export const createCommunity = async (req, res) => {
  const { name, code, description } = req.body;
  try {
    const communityExisted = await Community.findOne({ name });
    if (communityExisted) {
      return res.status(400).json({ success: false, message: "Community
already exists" })
    }
    const community = new Community({
      name,
      description,
      code,
      admin: req.userId,
      members: [req.userId]
    })

    const user = await User.findById(req.userId)
    user.createdCommunities.push(community._id)

    await user.save()
    await community.save()
    res.status(201).json({ success: true, message: "Commuunity created
successfully", community })
  } catch (error) {
    console.log(error, "failed to create community");
  }
}
```



```

        res.status(400).json({ success: false, message: "Failed to create
community" })

    }
}
export const joinCommunity = async (req, res) => {
    const { code } = req.body;

    try {
        if (!req.userId) {
            console.log("user not found");

            return res.status(400).json({ success: false, message: "User not
found" })
        }
        const community = await Community.findOne({ code })
        if (!community) {
            console.log("community not found");
            return res.status(400).json({ success: false, message: "Community
not Found" })
        }
        if (community.members.includes(req.userId)) {
            console.log("already a member");

            return res.status(400).json({ success: false, message: "Already a
member of this community" })
        }

        community.members.push(req.userId)
        await community.save()
    }
}

```

```

    console.log(community, "community joined successfully");

    const user = await User.findById(req.userId)
    user.communities.push(community._id)
    await user.save()

    res.status(200).json({ success: true, message: "Community joined
successfully" })

    } catch (error) {
        console.log(error, "failed to join community");
        res.status(400).json({ success: false, message: "Failed to join
community" })

    }

}

export const getJoinedCommunity = async (req, res) => {
    try {
        const joinedCommunity = await Community.find({ members:
req.userId }).select('name code admin ').populate('admin', 'name')
        console.log(joinedCommunity);

        res.status(200).json(joinedCommunity)

    } catch (error) {
        console.log(error, "failed to get the joined community");

```

```

        res.status(400).json({ success: false, message: "Failed to get the joined
community" })

    }

}

export const getCreatedCommunity = async (req, res) => {
    try {
        const createdCommunity = await Community.find({ admin: req.userId
}).select('name code admin description ').populate('admin', 'name email')
        console.log(createdCommunity);

        res.status(200).json(createdCommunity)

    }
    catch (error) {
        console.log(error, "failed to get the created community");
        res.status(400).json({ success: false, message: "Failed to get the created
community" })

    }
}

export const fetchCommunity = async (req, res) => {
    const { name } = req.params;
    try {
        const community = await Community.findOne({ name }).select('admin
name code description members').populate('members', 'name
email').populate('admin', 'name email')
        if (!community) {
            return res.status(400).json({ success: false, message: "Community
not found" })

```

```

    }
    res.status(200).json({ community, currentUserId: req.userId })

  } catch (error) {
    console.log(error, "failed to fetch community");
    res.status(400).json({ success: false, message: "Failed to fetch
community" })
  }
}

export const getMessages = async (req,res) => {
  const { name } = req.params;

  try {
    const community = await Community.findOne({ name
}).select("messages").populate("messages.user", "name email");

    if (!community) {
      return res.status(400).json({ success: false, message: "Community
not found" })
    }
    res.status(200).json({ community, currentUserId: req.userId })

  } catch (error) {
    console.log(error, "failed to fetch community");
    res.status(400).json({ success: false, message: "Failed to fetch
community" })
  }
}

```

```

export const sendMessages = async (req,res) => {
  const { code,message } = req.body;

  try {
    if (!req.userId) {
      console.log("user not found");

      return res.status(400).json({ success: false, message: "User not
found" })
    }
    const community = await Community.findOne({ code })
    if (!community) {
      console.log("community not found");
      return res.status(400).json({ success: false, message: "Community
not Found" })
    }

    const newMessage = { user: req.userId, message };
    community.messages.push(newMessage);
    await community.save();
    io.to(code).emit("recieveMessage", { user:req.userId, message });

    res.status(200).json({ success: true, message: "Message sent" })

  } catch (error) {
    res.status(400).json({ success: false, message: "Failed to send" })
  }
}

```

(Userprofilecontroller.js)

```
import { User } from "../models/userModel.js"
import jwt from 'jsonwebtoken'

export const updateUser=async(req,res)=>{
  const token =req.cookies.authToken;
  const {inputName,inputNumber}= req.body;
  if(!token){
    console.log("token is invalid");

  }
  try {
    const decoded= jwt.verify(token,process.env.JWT_SECRET_KEY)
    const userId=decoded.userId

    const user = await User.findByIdAndUpdate(userId,{
      name:inputName,
      number:inputNumber
    },{$set:req.body}, {new:true, runValidators:true})
    if(!user){
      res.status(400).json("user not found")
    }
    await user.save()
    res.status(200).json(user+" data updated successfully")

  } catch (error) {
    console.log(error+" failed updating user data");

    res.status(400).json("update failed")
  }
}
```

```

    }
  }

export const userProfile=async(req,res)=>{
  const token =req.cookies.authToken;

  if(!token){
    console.log("token is invalid ");

  }
  try {
    const decoded= jwt.verify(token,process.env.JWT_SECRET_KEY)
    const userId=decoded.userId;
    const user=await User.findById(userId)
    if(!user){
      res.status(400).json("user not found");
    }
    res.status(200).json(user);
  } catch (error) {
    console.log(error,"error finding user");

    res.status(400).json("user data is not available")
  }

}

```

(connectedDB.js)

```
import mongoose from "mongoose"

export const connectDb=async()=>{
  try {
    mongoose.connect(process.env.MONGO_URL)
    console.log("Mongodb Connected");

  } catch (error) {
    console.error(`Error connecting to MongoDB: ${error.message}`);
    process.exit(1);

  }
}
```

(Email.js)

```
import { PASSWORD_RESET_REQUEST_TEMPLATE,
PASSWORD_RESET_SUCCESS_TEMPLATE,
VERIFICATION_EMAIL_TEMPLATE } from "./mailTemplate.js";

import { sendEmail } from "./mail.js";

export const sendVerificationEmail = async (email, verificationToken) => {
  try {
    const htmlContent = VERIFICATION_EMAIL_TEMPLATE.replace(
      "{verificationCode}",
      verificationToken
    );
  }
```



```

    await sendEmail(email, "Verify Your Email", htmlContent);
    console.log("Verification email sent successfully");
  } catch (error) {
    console.log("Error sending verification email", error);
    throw new Error(`Error sending verification email: ${error}`);
  }
};

export const sendWelcomeEmail = async (name, email) => {
  const htmlContent = `

# Welcome ${name}!</h1><p>Thanks for joining CodCom!</p>`; try { await sendEmail(email, "Welcome to CodCom", htmlContent); console.log("Welcome email sent successfully"); } catch (error) { console.log("Error sending welcome email", error); throw new Error(`Error sending welcome email: ${error}`); } }; export const sendResetPasswordEmail = async(email,URL)={ const recipient = [{ email }]; try { const response = await mailtrapClient.send({ from: sender, to: recipient, subject: "Reset Password", html: PASSWORD_RESET_REQUEST_TEMPLATE.replace("{resetURL}",URL), category: "Password Reset", }); console.log("reset password email successfully", response); } catch (error) {


```

```

    console.log("error sending reset password email", error);
    throw new Error(`Error sending reset password email :${error}`);
  }

}

export const sendSuccessEmail=async(email)=>{
  const recipient = [{ email }];
  try {
    const response = await mailtrapClient.send({
      from: sender,
      to: recipient,
      subject: "Success",
      html: PASSWORD_RESET_SUCCESS_TEMPLATE,
      category: "Registration Success",
    });
    console.log("success email successfully", response);
  } catch (error) {
    console.log("error sending success email", error);
    throw new Error(`Error sending success email :${error}`);
  }
}

```

(mail.js)

```

import nodemailer from 'nodemailer';
import dotenv from 'dotenv';
dotenv.config();
// Set up transporter
const transporter = nodemailer.createTransport({
  service: 'gmail', // Replace with your email provider (Gmail, Outlook, etc.)
  auth: {

```

```

    user: process.env.MY_EMAIL, // Email from .env file
    pass: process.env.MY_EMAIL_PASS, // Password from .env file
  },
});

```

```

export const sender = process.env.MY_EMAIL;

```

```

export const sendEmail = async (recipient, subject, html) => {
  const mailOptions = {
    from: sender,
    to: recipient,
    subject,
    html,
  };

```

```

  try {
    const info = await transporter.sendMail(mailOptions);
    console.log('Email sent: ', info.response);
    return info;
  } catch (error) {
    console.error('Error sending email:', error);
    throw new Error('Error sending email');
  }
};

```

(mailtemplate.js)

```

export const VERIFICATION_EMAIL_TEMPLATE = `
<!DOCTYPE html>
<html lang="en">
<head>

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Verify Your Email</title>
</head>
<body style="font-family: Arial, sans-serif; line-height: 1.6; color: #333; max-
width: 600px; margin: 0 auto; padding: 20px;">
  <div style="background: linear-gradient(to right, #4CAF50, #45a049);
padding: 20px; text-align: center;">
    <h1 style="color: white; margin: 0;">Verify Your Email</h1>
  </div>
  <div style="background-color: #f9f9f9; padding: 20px; border-radius: 0 0 5px
5px; box-shadow: 0 2px 5px rgba(0,0,0,0.1);">
    <p>Hello,</p>
    <p>Thank you for signing up! Your verification code is:</p>
    <div style="text-align: center; margin: 30px 0;">
      <span style="font-size: 32px; font-weight: bold; letter-spacing: 5px; color:
#4CAF50;">{verificationCode}</span>
    </div>
    <p>Enter this code on the verification page to complete your registration.</p>
    <p>This code will expire in 15 minutes for security reasons.</p>
    <p>If you didn't create an account with us, please ignore this email.</p>
    <p>Best regards,<br>Your App Team</p>
  </div>
  <div style="text-align: center; margin-top: 20px; color: #888; font-size:
0.8em;">
    <p>This is an automated message, please do not reply to this email.</p>
  </div>
</body>
</html>
`;

```

```

export const PASSWORD_RESET_SUCCESS_TEMPLATE = `
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Password Reset Successful</title>
</head>
<body style="font-family: Arial, sans-serif; line-height: 1.6; color: #333; max-
width: 600px; margin: 0 auto; padding: 20px;">
  <div style="background: linear-gradient(to right, #4CAF50, #45a049);
padding: 20px; text-align: center;">
    <h1 style="color: white; margin: 0;">Password Reset Successful</h1>
  </div>
  <div style="background-color: #f9f9f9; padding: 20px; border-radius: 0 0 5px
5px; box-shadow: 0 2px 5px rgba(0,0,0,0.1);">
    <p>Hello,</p>
    <p>We're writing to confirm that your password has been successfully
reset.</p>
    <div style="text-align: center; margin: 30px 0;">
      <div style="background-color: #4CAF50; color: white; width: 50px; height:
50px; line-height: 50px; border-radius: 50%; display: inline-block; font-size:
30px;">
        ✓
      </div>
    </div>
    <p>If you did not initiate this password reset, please contact our support team
immediately.</p>
    <p>For security reasons, we recommend that you:</p>
    <ul>
      <li>Use a strong, unique password</li>

```

```

    <li>Enable two-factor authentication if available</li>
    <li>Avoid using the same password across multiple sites</li>
  </ul>
  <p>Thank you for helping us keep your account secure.</p>
  <p>Best regards,<br>Your App Team</p>
</div>
<div style="text-align: center; margin-top: 20px; color: #888; font-size:
0.8em;">
  <p>This is an automated message, please do not reply to this email.</p>
</div>
</body>
</html>
`;

```

```

export const PASSWORD_RESET_REQUEST_TEMPLATE = `
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Reset Your Password</title>
</head>
<body style="font-family: Arial, sans-serif; line-height: 1.6; color: #333; max-
width: 600px; margin: 0 auto; padding: 20px;">
  <div style="background: linear-gradient(to right, #4CAF50, #45a049);
padding: 20px; text-align: center;">
    <h1 style="color: white; margin: 0;">Password Reset</h1>
  </div>
  <div style="background-color: #f9f9f9; padding: 20px; border-radius: 0 0 5px
5px; box-shadow: 0 2px 5px rgba(0,0,0,0.1);">
    <p>Hello,</p>

```

```

    <p>We received a request to reset your password. If you didn't make this
request, please ignore this email.</p>
    <p>To reset your password, click the button below:</p>
    <div style="text-align: center; margin: 30px 0;">
        <a href="{resetURL}" style="background-color: #4CAF50; color: white;
padding: 12px 20px; text-decoration: none; border-radius: 5px; font-weight:
bold;">Reset Password</a>
    </div>
    <p>This link will expire in 1 hour for security reasons.</p>
    <p>Best regards,<br>Your App Team</p>
</div>
<div style="text-align: center; margin-top: 20px; color: #888; font-size:
0.8em;">
    <p>This is an automated message, please do not reply to this email.</p>
</div>
</body>
</html>
`;

```

(ismember.js)

```

import Community from '../models/communityModel.js';

export const isMember = async (req, res, next) => {
    const { communityId } = req.params;

    try {
        const community = await Community.findById(communityId);
        if (!community) {
            return res.status(404).json({ message: "Community not found" });
        }
    }

```

```

    if (!community.members.includes(req.user._id)) {
        return res.status(403).json({ message: "Access denied, you are not a
member of this community" });
    }

    next();
} catch (error) {
    res.status(500).json({ message: "Server error" });
}
};

```

(verifytoken.js)

```

import jwt from 'jsonwebtoken';
export const verifyToken=async(req,res,next)=>{
    const token=req.cookies.authToken;

    if(!token) {
        console.log("token not provided");

        return res.status(401).json({ success: false, message:"Token not provided" })
    }

    try {
        const decoded=jwt.verify(token,process.env.JWT_SECRET_KEY)
        if(!decoded) {
            return res.status(401).json({ success: false, message:"Token not
valid" })

```



```

    }
    req.userId=decoded.userId;
    next();

  } catch (error) {
    console.log("error verifying token", error);
    res.status(500).json({success: false, message:"Internal server error"})
  }
}

```

(communitymodel.js)

```

import mongoose from "mongoose";

const communitySchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
      unique: true,
    },
    description: {
      type: String,
      required: true,
    },
    code: {
      type: String,

```

```

    required: true,
    unique: true,
  },
  admin: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
  },
  members: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
    },
  ],
  messages: [
    {
      user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
      },
      message: String,
      createdAt: { type: Date, default: Date.now }

    }
  ],
},
{ timestamps: true }
);
const Community = mongoose.model("Community", communitySchema);
export default Community;

```

(usermodel.js)

```
import mongoose from "mongoose";
const userSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    number: { type: Number },
    communities: [{ type: mongoose.Schema.Types.ObjectId, ref: "Community"
  }],
    createdCommunities: [
      { type: mongoose.Schema.Types.ObjectId, ref: "Community" },
    ],
    lastLogin: { type: String, default: Date.now() },
    isVerified: { type: Boolean, default: false },
    verificationToken: String,
    verificationTokenExpiresAt: Date,
    resetToken: String,
    resetTokenExpiration: Date,
  },
  { timestamps: true }
);

export const User = mongoose.model("User", userSchema);
```

(codeAuth.js)

```
import express from 'express';
import { checkAuth, forgotPassword, login, logout, resetPassword, signup,
verifyEmail } from '../controller/codeAuthController.js';
import { verifyToken } from '../middleware/verifyToken.js';

const router= express.Router();
router.get('/check-Auth',verifyToken,checkAuth);

router.post('/signup',signup)
router.post('/login',login)
router.post('/logout',logout)
router.post('/verify-email',verifyEmail)
router.post('/forgot-password',forgotPassword)
router.post('/reset-password/:token',resetPassword)

export default router;
```

(communityroute.js)

```
import express from 'express';
import {
createCommunity, fetchCommunity, getCreatedCommunity, getJoinedCommuni
nity, joinCommunity, getMessages , sendMessages} from
'../controller/communityController.js';

import { verifyToken } from '../middleware/verifyToken.js';
```

```

const router = express.Router();

router.post('/createCommunity',verifyToken,createCommunity)
router.post('/joinCommunity',verifyToken, joinCommunity)
router.get('/getJoinedCommunity',verifyToken,getJoinedCommunity)
router.get('/getCreatedCommunity',verifyToken,getCreatedCommunity)
router.get('/:name',verifyToken,fetchCommunity)
router.get('/getMessages/:name',verifyToken,getMessages)
router.post('/sendMessages',verifyToken,sendMessages)
export default router;

```

(userroute.js)

```

import express from 'express';
import { updateUser, userProfile } from '../controller/userProfileController.js';

const router= express.Router()
router.get('/userProfile',userProfile)
router.post('/updateUser',updateUser)

export default router;

```

(generatetokenandsetcookies.js)

```

import jwt from 'jsonwebtoken'

export const generateTokenAndSetCookie=async(res,userId)=>{
  const token=jwt.sign({ userId },process.env.JWT_SECRET_KEY,{
    expiresIn:'7d'

```

```

    })
    res.cookie("authToken",token,{

        httpOnly:true,
        secure:process.env.NODE_ENV==='production'
        ||process.env.NODE_ENV==='development',
        sameSite:"none",
        maxAge: 7*24*60*60*1000
    })
    return token

}

```

(.env)

```

PORT=2024
MONGO_URL=mongodb+srv://vishwas_23a:wnSC74rtg4NGsYRt@mydaraba
se.jyzdgzw.mongodb.net/codeComAuth?retryWrites=true&w=majority&appNa
me=myDaraBase
JWT_SECRET_KEY=mysecretkey
NODE_ENV=development
MAILTRAP_TOKEN=9a98eb63279f6c7b1f531902a999b374
MY_EMAIL=codecom03@gmail.com
MY_EMAIL_PASS=yfimdlbjzbsxczk
RESET_URL=http://localhost:2024

```

(index.js)

```
import express from 'express';
import dotenv from 'dotenv';
import { createServer } from 'node:http';
import { Server } from "socket.io";
import authRoutes from './router/codeAuth.js'
import userRoute from './router/userRoute.js'
import communityRoute from './router/communityRoute.js'
import { connectDb } from './db/connectDb.js';
import cookieParser from 'cookie-parser';
import cors from 'cors'
import Community from './models/communityModel.js';
dotenv.config();
const port = process.env.PORT

const corsOptions = {
  origin: 'http://localhost:5173', // Replace with your frontend URL
  credentials: true,
  methods: ["GET", "POST", "PUT", "DELETE"],
  allowedHeaders: ["Content-Type", "Authorization"]
};

const app = express();
const server = createServer(app);
const io = new Server(server, {
  cors: {
    origin: 'http://localhost:5173',
    methods: ["GET", "POST", "PUT", "DELETE"],
    credentials: true
  }
});
```

```

app.use(cookieParser())
app.use(cors(corsOptions))
app.use(express.json())
app.use(express.urlencoded({ extended:true}))

app.use('/api/auth',authRoutes);
app.use('/api/user',userRoute)
app.use('/api/community',communityRoute)

const onlineUsers = new Map();
io.on('connection', (socket) => {
  console.log('a user connected');

  socket.on("joinCommunity", (communityCode) => {
    socket.join(communityCode);

    console.log(`User ${socket.id} joined community: ${communityCode}`);
  });

  socket.on("sendMessage", async ({ code, userId, message }) => {
    try {
      const community = await Community.findOne({ code });
      if (!community) return;

      const newMessage = { user: userId, message };
      community.messages.push(newMessage);
      await community.save();

      io.to(code).emit("receiveMessage", { user: userId, message });
    } catch (error) {
      console.error("Error handling sendMessage:", error);
    }
  });
});

```



```

    }
  });
  socket.on("disconnect", () => {
    console.log("A user disconnected:", socket.id);

  });
});
export default io;

server.listen(port,()=>{
  connectDb()
  console.log("server is live");

})

```

(package.json)

```

{
  "name": "backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "author": "vishwas",
  "type": "module",
  "license": "ISC",
  "description": "",
  "dependencies": {

```

```

    "bcryptjs": "^2.4.3",
    "cookie-parser": "^1.4.7",
    "cors": "^2.8.5",
    "crypto": "^1.0.1",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "jsonwebtoken": "^9.0.2",
    "mailtrap": "^3.4.0",
    "mongoose": "^8.13.0",
    "nodemailer": "^6.10.0",
    "nodemon": "^3.1.9",
    "socket.io": "^4.8.1"
  }
}

```

Frontend

(eslint.config.js)

```

import js from '@eslint/js'
import globals from 'globals'
import react from 'eslint-plugin-react'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'

export default [
  { ignores: ['dist'] },
  {

```

```

files: ['**/*.{js,jsx}'],
languageOptions: {
  ecmaVersion: 2020,
  globals: globals.browser,
  parserOptions: {
    ecmaVersion: 'latest',
    ecmaFeatures: { jsx: true },
    sourceType: 'module',
  },
},
settings: { react: { version: '18.3' } },
plugins: {
  react,
  'react-hooks': reactHooks,
  'react-refresh': reactRefresh,
},
rules: {
  ...js.configs.recommended.rules,
  ...react.configs.recommended.rules,
  ...react.configs['jsx-runtime'].rules,
  ...reactHooks.configs.recommended.rules,
  'react/jsx-no-target-blank': 'off',
  'react-refresh/only-export-components': [
    'warn',
    { allowConstantExport: true },
  ],
},
]

```

(index.html)

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>CodeCom</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

(package.json)

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
```

```

    "axios": "^1.8.4",
    "framer-motion": "^11.18.2",
    "motion": "^11.18.2",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-icons": "^5.5.0",
    "react-router-dom": "^7.4.0",
    "react-toastify": "^10.0.6",
    "socket.io-client": "^4.8.1"
  },
  "devDependencies": {
    "@eslint/js": "^9.15.0",
    "@types/react": "^18.3.12",
    "@types/react-dom": "^18.3.1",
    "@vitejs/plugin-react": "^4.3.4",
    "autoprefixer": "^10.4.20",
    "eslint": "^9.15.0",
    "eslint-plugin-react": "^7.37.2",
    "eslint-plugin-react-hooks": "^5.0.0",
    "eslint-plugin-react-refresh": "^0.4.14",
    "globals": "^15.12.0",
    "postcss": "^8.4.49",
    "tailwindcss": "^3.4.16",
    "vite": "^6.0.1"
  }
}

```

(postcss.config.js)

```

export default {
  plugins: {

```

```
tailwindcss: {},
autoprefixer: {},
},
}
```

(tailwind.config.js)

```
** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

(vite.config.js)

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

CHAPTER-5

SYSTEM DESIGN

System design is a solution, how to approach to the creation of a new system. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Emphasis is on translating the performance requirements into design specifications.

Design goes through logical and physical stages of development. Logical design reviews the present physical system; prepared input and output specifications; details the implementation plan; and prepares a logical design walkthrough. The physical design maps out the details of the physical system, plans the system implementation, devises a test and implementation plan, and specifies any new hardware and software.

5.1 Architecture

The architecture of Seva Sangh follows a client-server model, which is a common architectural pattern in web applications. In this model:

Client: The client-side of the application is responsible for presenting the user interface and handling user interactions. In the case of Seva Sangh, the client-side is implemented using React.js, a popular JavaScript library for building interactive user interfaces. React.js allows for the creation of dynamic, single-page applications (SPAs) where content is dynamically updated without requiring full page reloads.

Server: The server-side of the application manages the business logic, data processing, and communication with external services. Seva Sangh's server-side is built using JavaScript, a Java-based framework for building web applications. JavaScript simplifies the process of developing robust, production-ready web applications by providing out-of-the-box features such as dependency injection, security, and configuration.

The client and server communicate with each other using HTTP (Hypertext Transfer Protocol), a standard protocol for transmitting data over the web. The client sends requests to the server, which processes the requests, performs any necessary operations, and sends back responses containing the requested data.

The client-server architecture of Seva Sangh provides several benefits, including scalability, maintainability, and separation of concerns. By separating the user interface from the backend logic, the application becomes easier to maintain and scale as each component can be developed, tested, and deployed independently.

Architectural Design

Architectural design represents the structure of data and program components that are required to build a computer-based system. It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of a system.

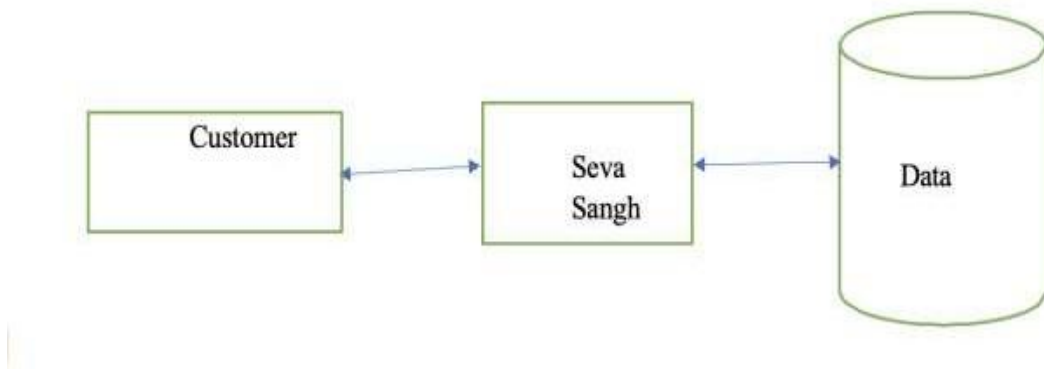


Fig5.1 Architectural Design

5.2 Database Structure:

The database structure of Seva Sangh is designed to efficiently store and manage various types of data required by the application. The application utilizes a relational database management system (RDBMS) to organize data into structured tables with predefined relationships.

Here's a breakdown of the main tables in the database:

User Table: This table stores information about registered users, such as user ID, username, email, password, registration date, and role. Each user is assigned a unique user ID, which serves as the primary key for the table.

Personal Table: This table stores additional personal information about users, such as their first name, last name, gender, birthdate, address, and phone number. The personal information is associated with the corresponding user through a foreign key relationship.

Education Table: This table stores educational background information about users, including their degree, school, major, and graduation date. Similar to the personal table, the education information is linked to the user through a foreign key relationship.

Other Table: This table serves as a placeholder for additional data that may be required by the application in the future. It can be customized to store various types of data based on the specific requirements of the application.

By organizing data into structured tables with defined relationships, the database structure ensures data integrity, consistency, and efficiency in storing and retrieving information.

5.3 Component Interactions:

The interaction between different components of Seva Sangh is crucial for the seamless operation of the application. Here's how the main components interact with each other:

User Interface (UI): The frontend user interface is responsible for presenting the application's features and allowing users to interact with the system. The UI is implemented using React.js components, which render dynamically based on user actions and data received from the server.

Backend Server: The backend server handles user requests, processes business logic, and interacts with the database and external services. It exposes RESTful APIs (Application Programming Interfaces) that the frontend client can communicate with to perform various actions such as user authentication, and updating user information.

External APIs: The application integrates with external APIs to retrieve real-time data. These APIs provide endpoints for accessing market data, historical

price information, and other relevant metrics. The backend server communicates with these APIs using HTTP requests to fetch the required data, which is then processed and sent back to the frontend client.

Database: The relational database stores persistent data required by the application, including user information. The backend server interacts with the database using SQL (Structured Query Language) queries to perform CRUD operations (Create, Read, Update, Delete) on the data.

Overall, the interaction between these components forms the backbone of Seva Sangh, allowing users to seamlessly monitor in real-time while ensuring data integrity, security, and performance. The client-server architecture, database structure, and component interactions are carefully designed.

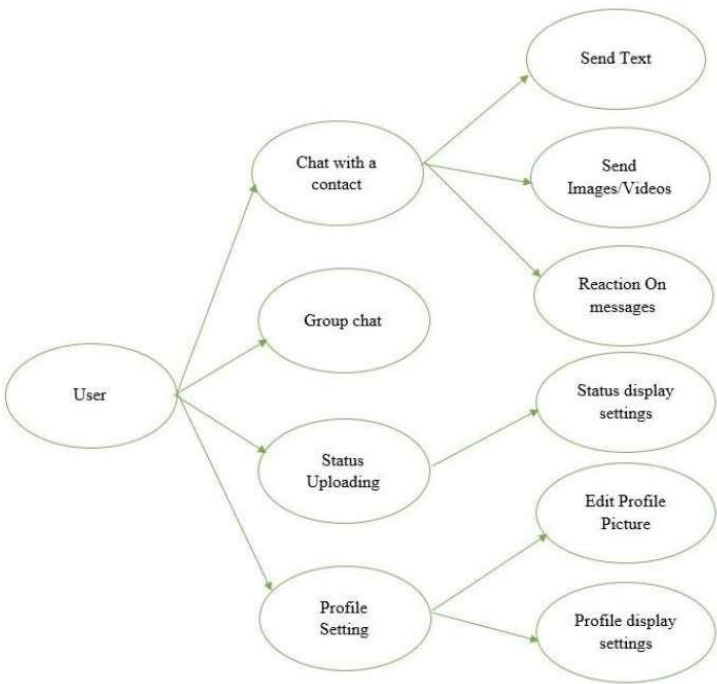


Fig5.3 Component Interactions

ARCHIETECTURE OF THE SYSTEM

An Architecture of a system in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks.

Its for higher level, less detailed descriptions that are intended to clarify overall concepts without concern for the details of implementation.

5.4 Control Flow Graph:

A Control Flow Graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications.

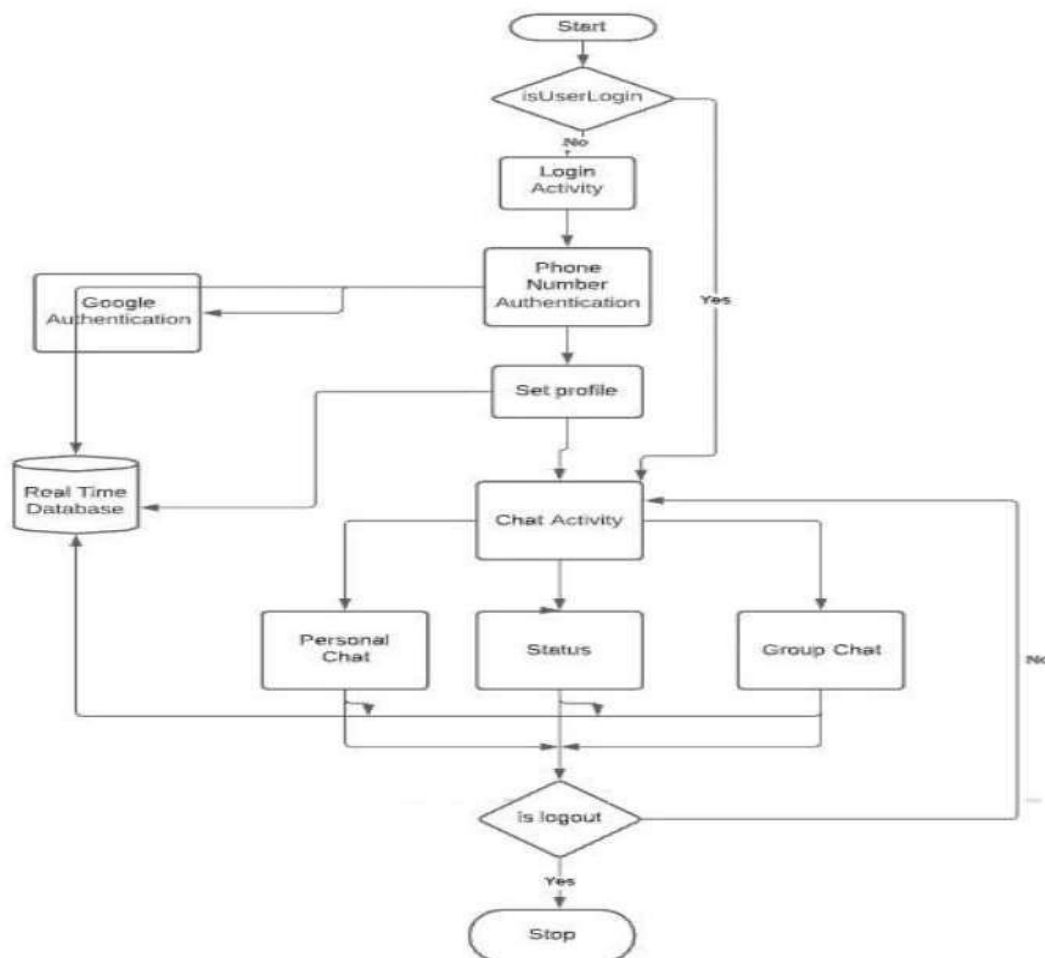


Fig5.4 Control Flow Graph

CHAPTER-6

DEPLOYMENT AND TESTING

6.1 Deployment on Vercel

Why Vercel ?

Vercel was chosen for deployment due to its ease of use, speed, and ability to handle modern web applications efficiently.

Implementation:

The Seva Sangh project was built and deployed on Vercel to ensure a fast, secure, and scalable hosting solution.

Vercel's automatic CI/CD (Continuous Integration/Continuous Deployment) pipeline allowed seamless updates by linking the project repository directly to the deployment process.

It provided features like automatic SSL certification, global content delivery through CDNs, and rollback capabilities for stable deployments.

1. Testing Objectives

The following are the testing objectives:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet-undiscovered error
- Successful test is one that uncovers an as yet undiscovered error.

2. Testing Principles

The basic principles that guide software testing are as follows:

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- The parate principle applies to software testing.

Pareto principle states that 80 percent of all errors uncovered during testing will likely betraceable to 20 percent of all program components.

Testing should begin —in the small —and progress toward testing —in the large. Exhaustive testing is not possible.

6.2 LEVEL OF TESTING

There are different levels of testing

- 1) Unit Testing
- 2) Integration Testing
- 3) System Testing

Integration testing

Integrating testing is a systematic technique for constructing Program structure while conducting tests to uncover error associates with interfacing. The objective isto take unit modules and built a program structure that has been directed by design.

- Integration Testing will test whether the modules work well together.
- This will check whether the design is correct.

System Testing

System testing is the process of testing the completed software as a part of the environment it was created for. It is done to ensure that all the requirements specified by the customer are met. System testing involves functional testing and performance testing.

System Testing will contain the following testing:

- 1) Functional Testing.
- 2) Performance Testing.

- Function Testing will test the implementation of the business needs.
- Performance Testing will test the non-functional requirements of the system like the speed, load etc.

6.3 Testing with Postman

Why Postman?

Postman is a powerful API testing tool that simplifies the process of testing backend endpoints, ensuring reliability and correctness.

Implementation:

All RESTful APIs were tested using Postman to verify their functionality, such as user authentication, data retrieval, and donation transactions.

Scenarios tested included:

Successful and failed login attempts.

CRUD (Create, Read, Update, Delete) operations on MongoDB for users and donations. Error handling for invalid or incomplete requests.

CHAPTER-7

INTENDED OUTCOMES

7.1 Functional Deliverables:

- A fully functional web application that connects individuals in need with donors and volunteers.
- Efficient, user-friendly interfaces for users to:
- Register as a donor or volunteer.
- Post requests for assistance.
- Match requests with available resources or volunteers.
- A robust database using MongoDB to manage user information and transactions securely.

7.2 Societal Impact:

- **Empowerment of Needy Individuals:** Facilitate timely assistance for individuals struggling with poverty or related challenges.
- **Increased Social Awareness:** Encourage users to participate actively in community welfare by donating resources or offering their time.
- **Stronger Community Bonds:** Foster a sense of collaboration and support among members of the community.

7.3 Scalability and Sustainability

- A scalable design that allows for future expansion, such as adding more categories of assistance (education, health, employment).

- Build a foundation for potential collaboration with NGOs or local organizations for long-term sustainability.

7.4 Educational and Professional Growth

- Hands-on experience with cutting-edge technologies like MongoDB and deployment platforms like Vercel.
- Deepened understanding of Agile methodology and its practical application in project management.
- Development of teamwork, problem-solving, and project planning skills.

7.5 Positive Recognition

- Showcase innovation and social responsibility at your college's annual events or tech fairs.
- Potential to inspire similar projects among peers, encouraging a culture of socially impactful technological solutions.

7.6 Validation and Testing Outcomes

- A thoroughly tested and optimized platform, validated using tools like Postman, ensuring a smooth user experience.
- Evidence of successful interactions (e.g., users matched with assistance providers, requests fulfilled).

7.7 Documentation and Learning

- Comprehensive documentation, including system architecture, user guides, and lessons learned.

CHAPTER-8

CHALLENGES AND RISKS

Identifying the challenges and risks for our Seva Sangh project is critical to ensure its success. Here are some potential challenges and risks, categorized into different areas:

8.1 Technical Challenges

- **Scalability Issues:** The application may face performance challenges as the user base grows, especially with a large MongoDB database.
- **Integration Problems:** Issues may arise in integrating various features such as matching algorithms, user notifications, and third-party services.
- **Testing Complexities:** Ensuring thorough testing for all edge cases using tools like Postman can be time-intensive and may still leave vulnerabilities.
- **Deployment Issues:** Hosting on platforms like Vercel might pose challenges in terms of configuration or resource limits for free tiers.

8.2 User Adoption Challenges

- **Awareness and Engagement:** Encouraging users to adopt the platform and actively participate as donors or volunteers might be challenging.
- **Trust Issues:** New users might hesitate to share personal information or resources due to concerns about data privacy or fraud.
- **Balancing Supply and Demand:** Ensuring there are enough donors and volunteers to meet the needs of users seeking help.

8.3 Data Management and Security Risks

- **Data Privacy:** Safeguarding sensitive user information from breaches or unauthorized access.
- **Data Accuracy:** Risks of inaccurate or incomplete user information impacting matching efficiency.
- **Fraudulent Activity:** Potential misuse of the platform, such as fake requests or malicious users.

8.4 Project Management Challenges

- **Time Constraints:** Managing deadlines and balancing academic responsibilities with project development.
- **Scope Creep:** The temptation to add new features beyond the planned scope could lead to delays.

8.5 Social and Ethical Risks

- **Unintended Bias:** Algorithms for matching requests with donors/volunteers might unintentionally favor certain users over others.
- **Ethical Concerns:** Ensuring fairness and transparency in resource allocation.
- **Dependency:** Avoiding a scenario where users become overly dependent on the platform without seeking other sustainable solutions.

8.6 External Risks

- **Internet Access:** Users with limited internet connectivity may face difficulties in accessing the platform.

- **Competition:** Other similar platforms or NGOs might provide more established solutions, reducing your user base.

8.7 Risk Mitigation Strategies

To address these challenges:

- **Technical Solutions:** Implement robust testing frameworks, scalable architecture, and secure coding practices.
- **Awareness Campaigns:** Create promotional content to increase awareness and user trust.
- **Monitoring and Reporting:** Develop mechanisms for reporting misuse or fraudulent activity.
- **Team Collaboration:** Use project management tools to maintain clear communication and task tracking.
- **Ethical Guidelines:** Establish transparent policies for fairness and data use.

CHAPTER-9

CONCLUSION

Seva Sangh stands as a testament to how technology can be harnessed to address pressing societal challenges. Designed as a college initiative, the project aimed to create a web application that connects individuals in need with donors and volunteers, fostering a sense of community and mutual support. By leveraging modern technologies, collaborative teamwork, and innovative problem-solving, Seva Sangh provides a platform that embodies the principles of empathy, efficiency, and empowerment.

9.1 Technical Achievements

The project showcased the successful integration of various advanced tools and methodologies:

1. **Backend Infrastructure:** MongoDB provided a robust and scalable database solution to handle user information and transactions, ensuring reliability and efficiency.
2. **Deployment Excellence:** Hosting the application on Vercel enabled a seamless and accessible user experience.
3. **Development Process:** Employing the Agile methodology facilitated iterative development, adaptability to changing requirements, and continuous improvement.
4. **Testing and Validation:** Postman was used extensively to test the API endpoints, ensuring the platform was free from major bugs and delivered a smooth user experience.

9.2 Social Impact

The primary vision of Seva Sangh was to combat poverty and promote inclusivity by connecting donors and volunteers with individuals in need. The platform achieved this through:

1. **Empowering the Needy:** By enabling users to post their requests, the project bridged the gap between those seeking help and those willing to provide it.
2. **Fostering Generosity:** The application encouraged donors and volunteers to actively contribute to society, enhancing social cohesion and responsibility.
3. **Creating Awareness:** Highlighting the importance of collaboration in addressing socio-economic disparities, Seva Sangh inspired its users to make meaningful contributions.

9.3 Challenges Addressed

Throughout its development, the project overcame significant challenges:

- Balancing the supply of donors/volunteers with the demand from users in need.
- Ensuring data privacy and security, particularly for sensitive user information.
- Building trust among users to encourage participation and engagement.
- Managing time constraints and academic responsibilities alongside the project.

By addressing these challenges effectively, the team ensured the platform was both practical and impactful.

9.4 Learning Outcomes

The project provided invaluable learning experiences for the team:

- **Technical Expertise:** Enhanced proficiency in web development, database management, and deployment tools.

- **Collaboration:** Strengthened teamwork, communication, and project management skills.
- **Real-World Application:** A deeper understanding of how to design technology-driven solutions for real-world problems.
- **Critical Thinking:** Tackling unforeseen challenges with creativity and problem-solving abilities.

9.5 Future Scope

While Seva Sangh has achieved its initial objectives, its potential for future growth remains vast:

- **Feature Expansion:** Incorporating additional categories like educational support, healthcare, or job assistance.
- **Partnerships:** Collaborating with NGOs, government bodies, or local communities to amplify its reach.
- **AI Integration:** Leveraging AI to improve matching algorithms, ensuring more accurate and efficient pairing of requests and resources.
- **Mobile Application:** Developing a mobile app to increase accessibility for users with limited internet connectivity.

9.6 Final Reflection

Seva Sangh is more than just a college project; it is a platform that embodies the spirit of service and the power of technology to create meaningful change. By bridging gaps and fostering connections, the project stands as a symbol of what can be achieved when innovation meets social responsibility. It leaves behind not only a functional application but also a vision of hope, collaboration, and a better future for all.

CHAPTER-10

REFERENCES

1. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin

This book provides essential principles and practices for writing clean, maintainable code—

perfect for developing a robust application.

2. MDN Web Docs

A comprehensive resource for web development, covering everything from HTML and CSS to JavaScript and frameworks like Next.js.

<https://developer.mozilla.org/en-US/blog/static-site-generation-with-nextjs/>

3. FreeCodeCamp

Offers tutorials and practical guides for full-stack development, including MongoDB, Next.js, and Agile methodologies.

<https://www.freecodecamp.org/news/react-context-api-tutorial-examples/>

4. Postman Learning Center

For improving API development and testing skills, which is crucial for ensuring seamless interactions between users and systems in your application.

<https://learning.postman.com/docs/getting-started/first-steps/sending-the-first-request/>

5. "Designing Data-Intensive Applications" by Martin Kleppmann

This book provides a deep dive into building scalable, reliable, and maintainable systems— ideal for the backend and data integration aspects of your project.

6. Next.js Official Documentation

A comprehensive guide for implementing Next.js features like SSR (Server-Side Rendering) and API routes, critical for building dynamic web platforms.

7. Prismic Blog on Next.js Examples

A curated list of Next.js project examples with detailed instructions, including e-commerce and dynamic content rendering projects, to inspire your project development.

8. MongoDB Developer Center

Offers resources and guides on integrating MongoDB with Next.js, suitable for creating a scalable database backend for donation tracking.

<https://www.mongodb.com/developer/quickstart/nextjs/>