BSSE FINAL PROJECT
Design and Test Specification

CodeFlow

Project Advisor

Mohsin Sami

Presented by:
Group ID: F23SE052

Student Reg#	Student Name
L1F20BSSE0191	Muhammad Haseeb Nawaz
L1F20BSSE0183	Muhammad Mujeeb
L1F19BSSE0073	Rohan Qamar

Faculty of Information Technology
University of Central Punjab

Design and Test Specification

SDP Phase III

CodeFlow

Advisor: Mohsin Sami

Team F23SE052

Member Name	Primary Responsibility
Muhammad Haseeb Nawaz	**Requirement Specification, Implementation and documentation**
Muhammad Mujeeb	**Requirement Specification, Implementation and documentation**
Rohan Qamar	**Requirement Specification, Implementation and documentation**

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

Abstract

CodeFlow addresses the significant challenge faced by novice programmers in understanding abstract programming concepts and constructing logical program flows. This web-based platform provides an intuitive drag-and-drop interface for creating program code using flowchart elements. The core innovation lies in its functionality to convert flowcharts into textual code and vice versa.

By enabling step-by-step flowchart execution with memory map visualization, CodeFlow allows learners to observe data storage and manipulation during program execution. This dynamic and interactive learning environment simplifies programming complexities, making fundamental concepts more accessible. The project utilizes knowledge areas such as visual programming, algorithm design, and educational technology.

The anticipated results include improved comprehension of programming principles, enhanced problem-solving skills, and increased confidence among novice programmers. CodeFlow stands as a transformative tool in programming education, bridging the gap between visual and textual programming to foster a deeper understanding and retention of coding concepts.

1. Introduction

The field of programming education faces significant challenges in providing an accessible and effective learning environment for novice programmers and students. The abstract nature of code, combined with the complexities involved in creating logical program flows, poses substantial barriers to understanding fundamental programming concepts. Aspiring programmers often struggle to comprehend the workings behind the scenes, such as memory allocation and constructing effective logical structures. The absence of user-friendly tools exacerbates these issues, hindering learning progress and stifling enthusiasm for programming.

CodeFlow addresses these challenges by providing a web-based platform with a user-friendly drag-and-drop interface. This platform simplifies the complexities of logical structure creation by allowing users to construct program code using flowchart elements. It incorporates a accurate flowchart-to-code and code-to-flowchart conversion, ensuring a seamless transition from visual representations to textual code. Additionally, CodeFlow integrates memory map visualization, enabling learners to observe data storage and manipulation during program execution.

By offering a dynamic and interactive learning environment, CodeFlow aims to empower novice programmers, making programming concepts more accessible and comprehensible. This project is designed to foster confidence, enhance problem-solving skills, and encourage creativity among learners. Ultimately, CodeFlow aspires to revolutionize programming education by providing a comprehensive and intuitive tool that transforms the learning experience for aspiring programmers and students.

1.1 Product

CodeFlow is an innovative web-based platform designed to aid novice programmers in understanding and constructing logical program flows through a drag-and-drop interface. It allows users to create program code using flowchart elements and provides accurate conversions between flowcharts and textual code.

The primary problem addressed by CodeFlow is the difficulty novice programmers face in comprehending abstract programming concepts and creating logical program structures. Traditional programming education methods often lack intuitive tools for visualizing code, making it challenging for beginners to grasp how code translates into logical flows and memory operations.

The end product will be a comprehensive educational tool that serves as a bridge between visual and textual programming. CodeFlow provides an interactive learning environment with features such as:

- Drag-and-drop flowchart creation.
- Accurate flowchart-to-code and code-to-flowchart conversion.
- Step-by-step program execution with memory map visualization.

This tool aims to enhance the programming learning experience by making abstract concepts more tangible and understandable, thereby fostering improved comprehension and problem-solving skills among novice programmers.

1.2 Background

In the domain of programming education, various tools and platforms have been developed to help novice programmers understand coding concepts. Notable among these are Code and Flow, Raptor, and Scratch. These tools have contributed significantly to programming education but come with limitations that CodeFlow aims to address.

1. Raptor:

- **Overview:** A flowchart-based programming environment designed to help beginners visualize their code.
- **Limitations:** Similar to Code and Flow, Raptor has limitations in handling complex logic and translating it effectively into visual form.
- **Comparison:** CodeFlow differentiates itself by providing real-time memory map visualization, allowing learners to see how data is stored and manipulated during execution, which Raptor does not offer.

2. Scratch:

- **Overview:** A visual programming language developed by MIT that introduces programming concepts through block-based coding.
- **Limitations:** While effective for younger audiences and basic programming concepts, Scratch lacks depth in transitioning to text-based programming languages.
- **Comparison:** CodeFlow goes beyond basic visual programming by offering a platform that not only introduces concepts visually but also transitions smoothly into textual code, making it suitable for learners advancing to more complex programming.

3. Code and Flow:

- Overview: A tool that translates visual flowcharts into textual code and vice versa.
- Limitations: It struggles with accurately convert flowcharts into textual code and vice versa.
- Comparison: CodeFlow enhances this functionality for accurate conversion, ensuring a seamless transition between visual and textual representations.

Differentiation from Other Tools

CodeFlow stands out from these tools in several ways:

- **Accurate Conversion**: It employs precise flowchart-to-code and code-to-flowchart conversions, ensuring learners can trust the visual representations.
- **Memory Map Visualization:** This feature allows users to see real-time data storage and manipulation, providing deeper insights into how programs execute.
- **Interactive Learning Environment:** The step-by-step execution of programs helps learners identify and understand errors in logic, enhancing their debugging skills.

By addressing the limitations of existing tools and incorporating advanced features, CodeFlow aims to provide a more effective and engaging learning experience for novice programmers, setting it apart from previous projects and current tools in the market.

1.3 Objective(s)/Aim(s)/Target(s)

1. Develop an Intuitive Interface: Create a user-friendly drag-and-drop interface for constructing program flowcharts.

2. Accurate Conversion: Implement precise conversion between flowcharts and textual code.

3. Step-by-Step Execution with Visualization: Integrate functionality for step-by-step program execution with memory map visualization.

4. Real-Time Feedback: Provide immediate insights and feedback during user interaction.

5. Browser Compatibility and Device Optimization: Ensure compatibility with major web browsers and optimize for various devices .

1.4 Scope

The scope of the CodeFlow project encompasses the development and implementation of a comprehensive web-based platform aimed at enhancing the learning experience for novice programmers and students.

1.5 Business Goals

This project does not have formal business goals or revenue targets. Its primary focus is on facilitating student learning.

1.6 Document Conventions

Font: Times
Body
Body size: 12
Line spacing: 0.5
Heading
Heading1 size: 18
Heading2 size: 14
Sub-Heading: 12
Line spacing: 1.5

2. Technical Architecture

1. Is the system custom-built or COTS?
    - CodeFlow is a custom-built system, incorporating the React Flow library and GDB as integral components of its unique architecture.

2. What type of processing is the current system responsible for?
    - The CodeFlow system primarily handles online processing with a focus on transaction processing. It is designed to interactively process user inputs such as flowchart creation and code generation in real-time, providing immediate feedback and results.

3. What are the major application components?
    - **Frontend Interface (React):** Manages user interactions, flowchart creation, and display, providing the drag-and-drop interface for building and editing flowcharts.
    - **Backend (Node.js):** Handles the logic for converting flowcharts to code and vice versa, managing GDB sessions for step-by-step code execution, and processing API requests from the frontend.
    - **GDB Integration:** Facilitates the step-by-step execution of the generated C++ code, allowing users to observe the flow of execution in correspondence with the flowchart.
    - **API:** Serves as the communication bridge between the frontend and backend, handling data transfer and command execution requests.
    - **Local Storage Management:** Manages saving and retrieving user data, flowcharts, and code snippets locally, as there is no database integration.

4. What data does the current system collect and manage?
    - The system does not manage persistent user data or profiles as it does not integrate a database.

5. What is the basic application architecture?
    - The basic application architecture of CodeFlow is a client-server model.

6. What programming languages is the current system built in?
    - **JavaScript (with React):** Used for developing the frontend interface, including the integration of the React Flow library for flowchart functionality.
    - **Node.js:** Used for the backend server development, especially considering the integration with a JavaScript-based frontend.
    - **C++:** Utilized in the context of GDB for the generation and step-by-step execution of C++ code from flowcharts.

7. What is the hardware platform that supports the current system?
    - The CodeFlow system is web-based, which means it primarily relies on the user's device with a modern web browser to access and interact with the platform. There are

no specific hardware requirements for the user's device beyond having internet access and a compatible web browser.

8. What database platform supports the current system?
    - CodeFlow does not rely on a traditional database platform for data storage. Instead, it utilizes JSON files for saving and exporting project data. Therefore, there is no specific database platform supporting the current system, as it operates without a database backend.

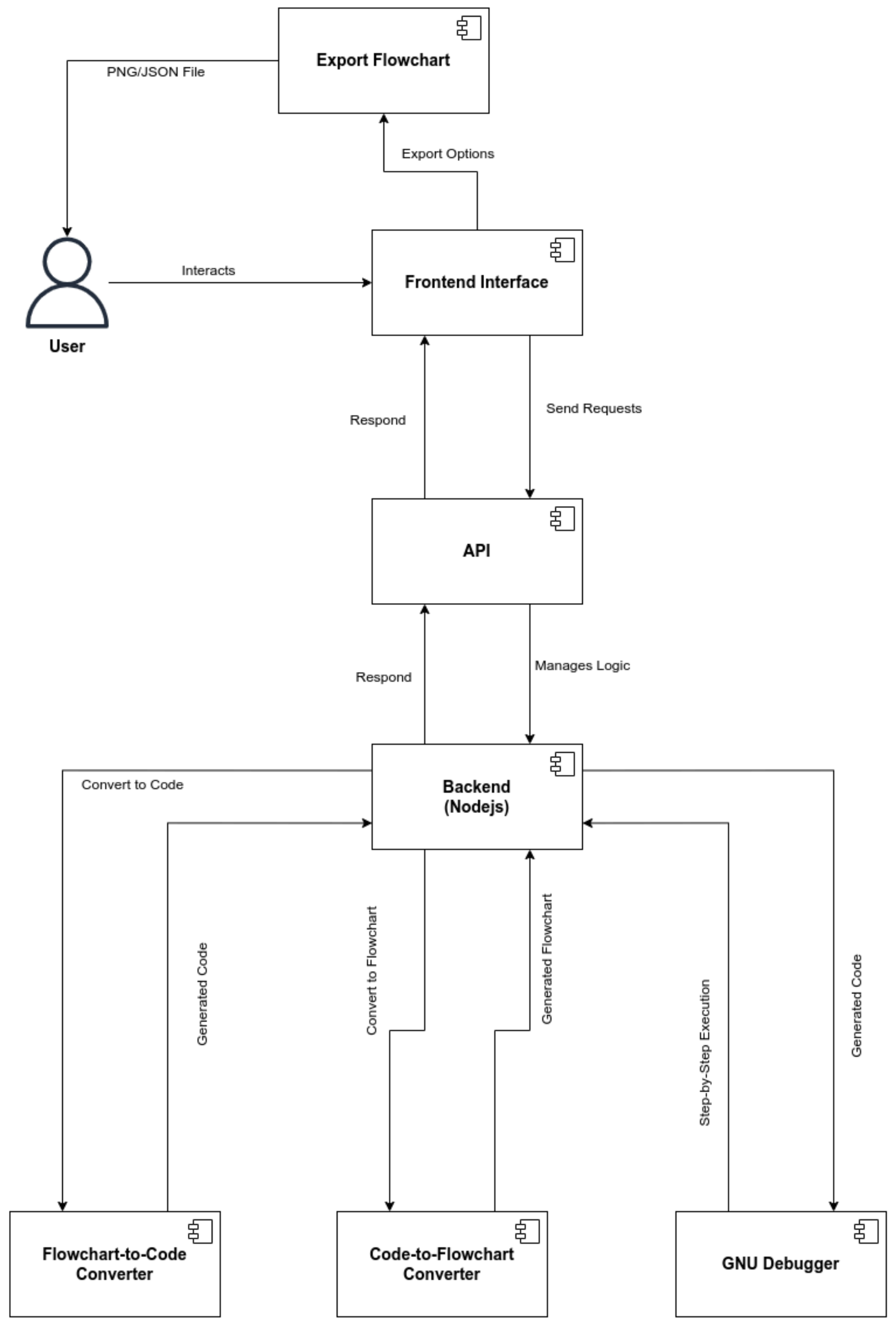
9. Does the system have an end-user interface? If so, what type of user interface?
    - Yes, the CodeFlow system has an end-user interface. The user interface is browser-based, which means it is accessed and interacted with through a web browser on the user's device.

10. What is the basic network architecture?
    - The basic network architecture for CodeFlow is designed to be accessible over the Internet. It is a web-based platform that users can access from any location with an internet connection.

11. Where is the system hosted?
    - The CodeFlow system is not hosted in a specific location. Instead, it is an open-source project that can be self-hosted by users. This means that users have the flexibility to deploy and run the application on their own computers according to their preferences and needs. The CodeFlow codebase and resources will be available for users to set up and host independently.

The CodeFlow system comprises several major components: the application components include the frontend interface built with React, which manages user interactions, flowchart creation, and display; the backend developed with Node.js, which handles the logic for converting flowcharts to code and vice versa, manages GDB sessions for step-by-step code execution, and processes API requests; and the GDB integration, which facilitates the execution of generated C++ code. Data components are managed through local storage, utilizing JSON files for saving and retrieving user data, flowcharts, and code snippets, as there is no traditional database integration. The API serves as the interfacing system, acting as a communication bridge between the frontend and backend, handling data transfer and command execution requests. Collaboration between these components is facilitated through a client-server model where the frontend sends user input to the backend via API calls, and the backend processes this data, executes necessary logic, and returns results to the frontend. The design employs the React Flow library for building the flowchart interface, showcasing design reuse. Tools and technologies used include JavaScript with React for the frontend, Node.js for the backend, and C++ in conjunction with GDB for code execution and debugging. This integration ensures a cohesive and interactive user experience while maintaining real-time processing and feedback.

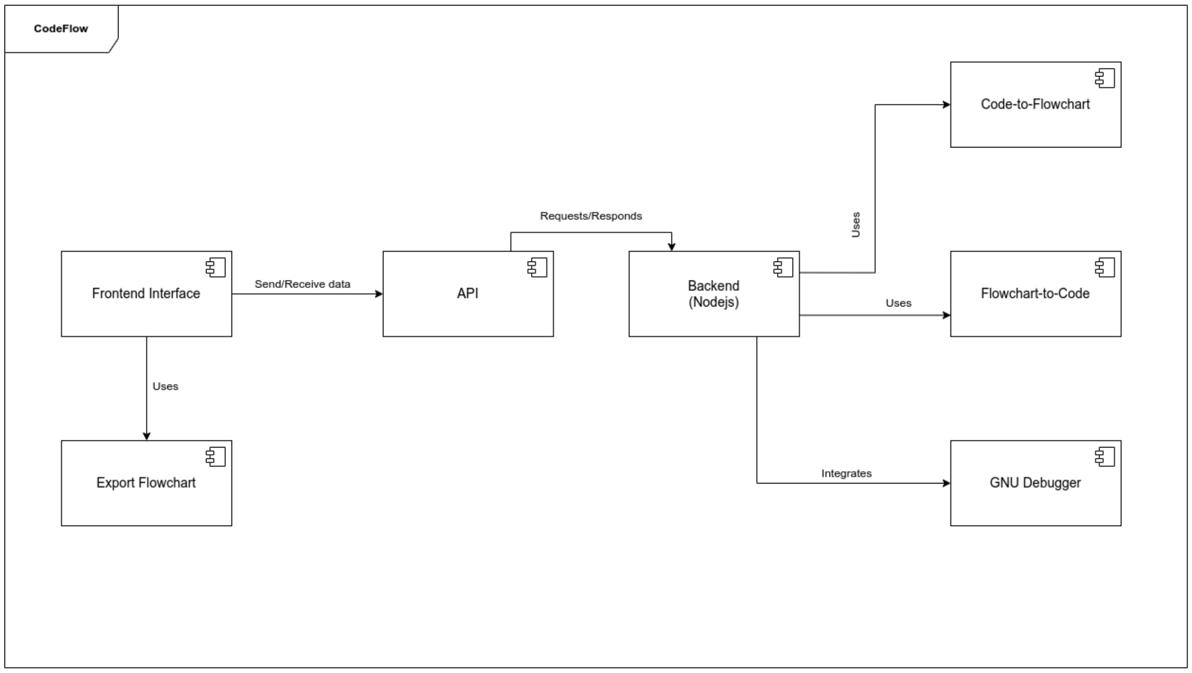
2.1 Application and Data Architecture



The architecture of CodeFlow consists of several key application components and processing units. The **User** is the primary actor who interacts with the system, utilizing various features provided by the application. The **Frontend Interface** is where all user interactions take place, allowing users to create, edit, and interact with flowcharts through a drag-and-drop interface. The **Export Flowchart** feature enables users to export their created flowcharts as PNG images or JSON files for future use or editing. The **API** acts as an intermediary, facilitating communication between the frontend interface and the backend server by transmitting user requests to the backend and delivering responses back to the frontend.

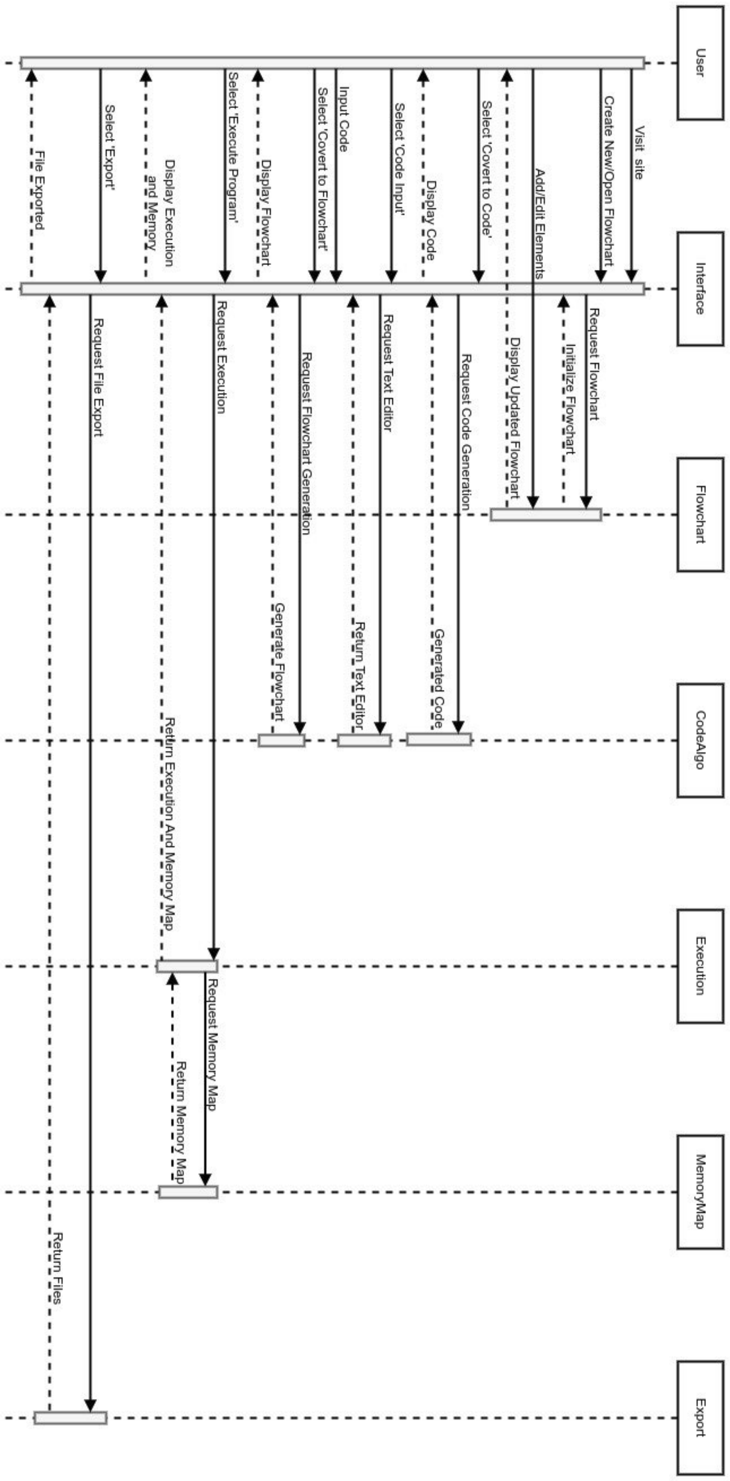
The **Backend (Node.js)** is the server-side component that handles the core logic of the application, including data processing and the execution of functionalities such as conversion algorithms and debugging processes. The **Flowchart-to-Code Converter** component converts user-created flowcharts into executable code, likely in C++, providing a seamless transition from visual to textual programming. Conversely, the **Code-to-Flowchart Converter** takes existing code and converts it into a visual flowchart representation, aiding users in understanding the code structure. The **GNU Debugger** is responsible for running the generated code step-by-step and providing insights into the program's memory usage. This includes allowing users to inspect variable values, the call stack, and memory consumption during execution, thereby enhancing their understanding of the program’s behavior. Together, these components create a cohesive system that simplifies programming education for novice learners.

Component Diagram:

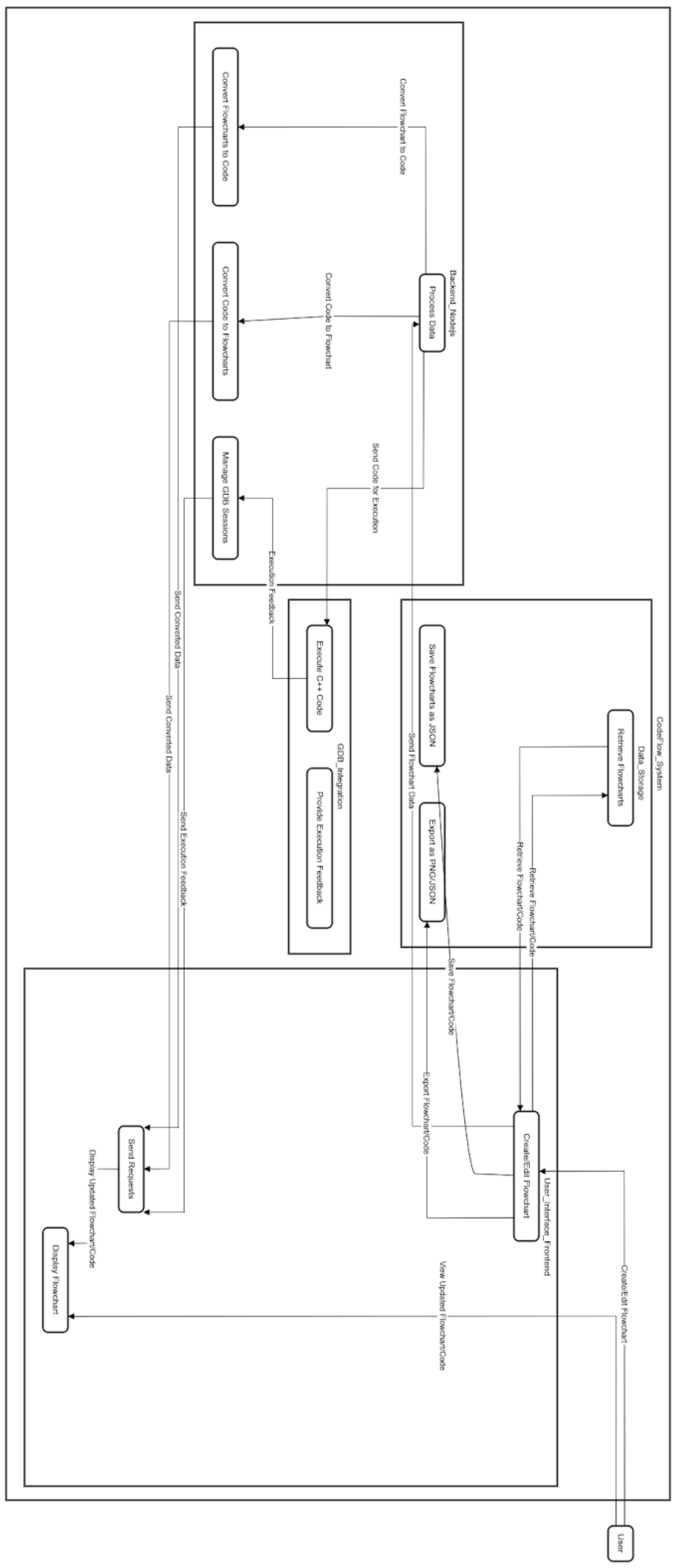


2.2 Component Interactions and Collaborations

Sequence Diagram:



Data Flow Diagram:



2.3 Design Reuse and Design Patterns

ReactFlow Library: The utilization of this third-party library for building the flowchart interface is a significant example of reuse. It provides some basic components and functionalities that save development time and ensure a reliable user experience.
GDB Integration: Using the GNU Debugger for step-by-step code execution and memory map is an example of tool reuse, as it leverages an existing, robust solution for debugging.

2.4 Technology Architecture

The anticipated infrastructure to support CodeFlow will involve a cloud-based, serverless architecture designed to minimize the need for ongoing infrastructure maintenance. This high-level technology architecture includes several key components:

1. Platform:

- **Frontend Hosting:** The frontend of CodeFlow will be hosted on a managed hosting service optimized for React applications. Services like Vercel or Netlify can be used to deploy and manage the frontend, ensuring high availability and performance.
- **Backend Services:** The backend logic will be implemented using Function as a Service (FaaS) platforms such as AWS Lambda or Google Cloud Functions. These serverless functions will handle flowchart-to-code conversion, code execution with GDB, and other computational tasks.

2. System Hosting:

- **Managed Hosting Services:** The frontend will be hosted on platforms that provide built-in support for static site generation and server-side rendering, ensuring efficient delivery of content.
- **Containerized Microservices:** For tasks requiring persistent computation, such as GDB sessions, containerized microservices (e.g., AWS Fargate or Google Cloud Run) will be used to run stateless containers.

3. Modes of Operation:

- **Online Mode:** CodeFlow operates primarily in an online mode, where all user interactions and processing tasks are performed in real-time over the Internet.
- **Self-Hosting:** The system will be designed as an open-source project, allowing users to self-host the application on their own servers if preferred. This flexibility ensures that users can customize and deploy the system according to their specific needs.

This infrastructure will provide a robust and scalable environment for CodeFlow, ensuring that it can handle user demands efficiently while providing a seamless and interactive learning experience.

2.5 Architecture Evaluation

1. Frontend Framework: React

- **Reasons for Selection:**
  - **Popularity and Community Support:** React is one of the most popular frontend frameworks with a large community, extensive documentation, and numerous libraries and tools.
  - **Component-Based Architecture:** React's component-based architecture allows for modular development, making the code more manageable and reusable.
  - **Performance:** React is known for its performance optimizations and efficient rendering with the virtual DOM.

- **Pros:**
  - High performance and fast rendering.
  - Large ecosystem and community support.
  - Easy to create reusable UI components.
  - Strong support for state management libraries like Redux.

- **Cons:**
  - Learning curve for beginners.
  - Rapidly evolving, which may require frequent updates to keep up with best practices.

- **Alternative: Vue.js**
- **Pros:**
  - Easier learning curve compared to React.
  - Flexible and integrates well with existing projects.
- **Cons:**
  - Smaller community compared to React.
  - Fewer third-party libraries and tools.

2. Backend Services: Node.js and Serverless Functions (AWS Lambda/Google Cloud Functions)

- **Reasons for Selection:**
  - **Event-Driven and Non-Blocking I/O:** Node.js is well-suited for real-time applications due to its asynchronous, non-blocking I/O.

- **Serverless Architecture:** Serverless functions allow for scalable, cost-effective backend processing without the need for managing servers.
- **JavaScript/Node.js Integration:** Using JavaScript for both frontend and backend simplifies development and enables code reuse.

- **Pros:**
  - Scalability and cost efficiency with serverless functions.
  - Single language (JavaScript) across the stack.
  - High performance for I/O-bound tasks.

- **Cons:**
  - Cold start latency for serverless functions.
  - Limited execution time for serverless functions.

- **Alternative: Traditional Server-Based Architecture (Express.js on Node.js)**
- **Pros:**
  - More control over server configuration and environment.
  - No cold start latency issues.
- **Cons:**
  - Requires managing and scaling servers.
  - Higher cost and maintenance overhead.

3. Containerization: AWS Fargate/Google Cloud Run

- **Reasons for Selection:**
  - **Ease of Deployment:** Containers simplify the deployment process by packaging the application and its dependencies together.
  - **Scalability:** Container orchestration services like AWS Fargate and Google Cloud Run handle scaling automatically.
  - **Isolation:** Containers provide a consistent runtime environment, reducing issues caused by differences in development and production environments.

- **Pros:**
  - Simplified deployment and scaling.
  - Consistent and isolated runtime environments.
  - Managed services reduce operational overhead.

- **Cons:**
  - Initial learning curve for containerization and orchestration.
  - Potential for over-reliance on vendor-specific features (vendor lock-in).

- **Alternative: Virtual Machines (VMs)**
- **Pros:**

  - Greater control over the environment.
  - No vendor lock-in.
- **Cons:**
  - Higher overhead in terms of resource usage and management.
  - Less efficient scaling compared to containers.

4. Storage: JSON Files for Local Storage

- **Reasons for Selection:**
  - **Simplicity:** Using JSON files for storage is straightforward and fits the needs of the application without requiring a full-fledged database.
  - **Flexibility:** JSON is a flexible format that can easily be read and written by JavaScript, the primary language used in CodeFlow.

- **Pros:**
  - Easy to implement and manage.
  - No need for complex database setup and maintenance.
  - Well-suited for small-scale, local data storage needs.

- **Cons:**
  - Not suitable for large-scale data or complex queries.
  - Lack of built-in data validation and indexing.

- **Alternative: NoSQL Database (e.g., MongoDB)**
- **Pros**:
  - Scalable and supports complex queries.
  - Built-in features for data validation and indexing.
- **Cons**:
  - More complex setup and maintenance.
  - Overhead may be unnecessary for small-scale projects.

Overall, the selected infrastructure and technologies are chosen for their balance of simplicity, performance, scalability, and community support. Alternatives were considered, but the selected technologies best fit the project requirements and team's expertise.

3. Detailed/Component Design

1. User Interactions:

- Users interact with the system primarily through the frontend interface to create, edit, save, retrieve, and export flowcharts.
- The frontend interface needs to provide a seamless user experience, handling user inputs efficiently and providing immediate feedback.

2. Frontend-Backend Interactions:

- The frontend sends user requests (e.g., flowchart creation, code conversion) to the backend via API calls.
- The backend processes these requests and sends responses back to the frontend to update the user interface.

3. Backend-External Tool Interactions:

- The backend integrates with GDB for code execution, sending the generated C++ code for step-by-step execution and receiving feedback on the execution process.

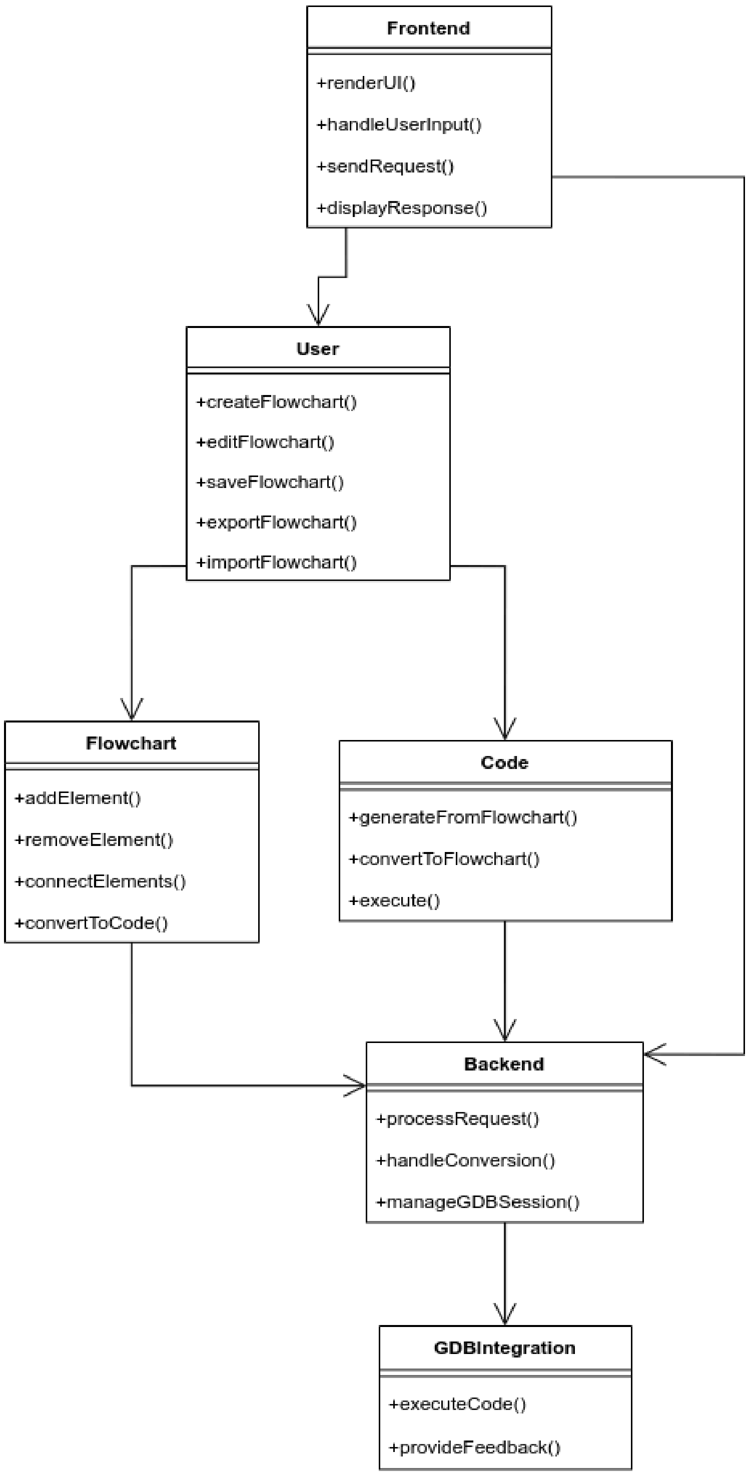
4. Data Storage Interactions:

- The application exports flowcharts as PNG or JSON files, which the user can download to their local machine.
- Users can import JSON files to resume their work.

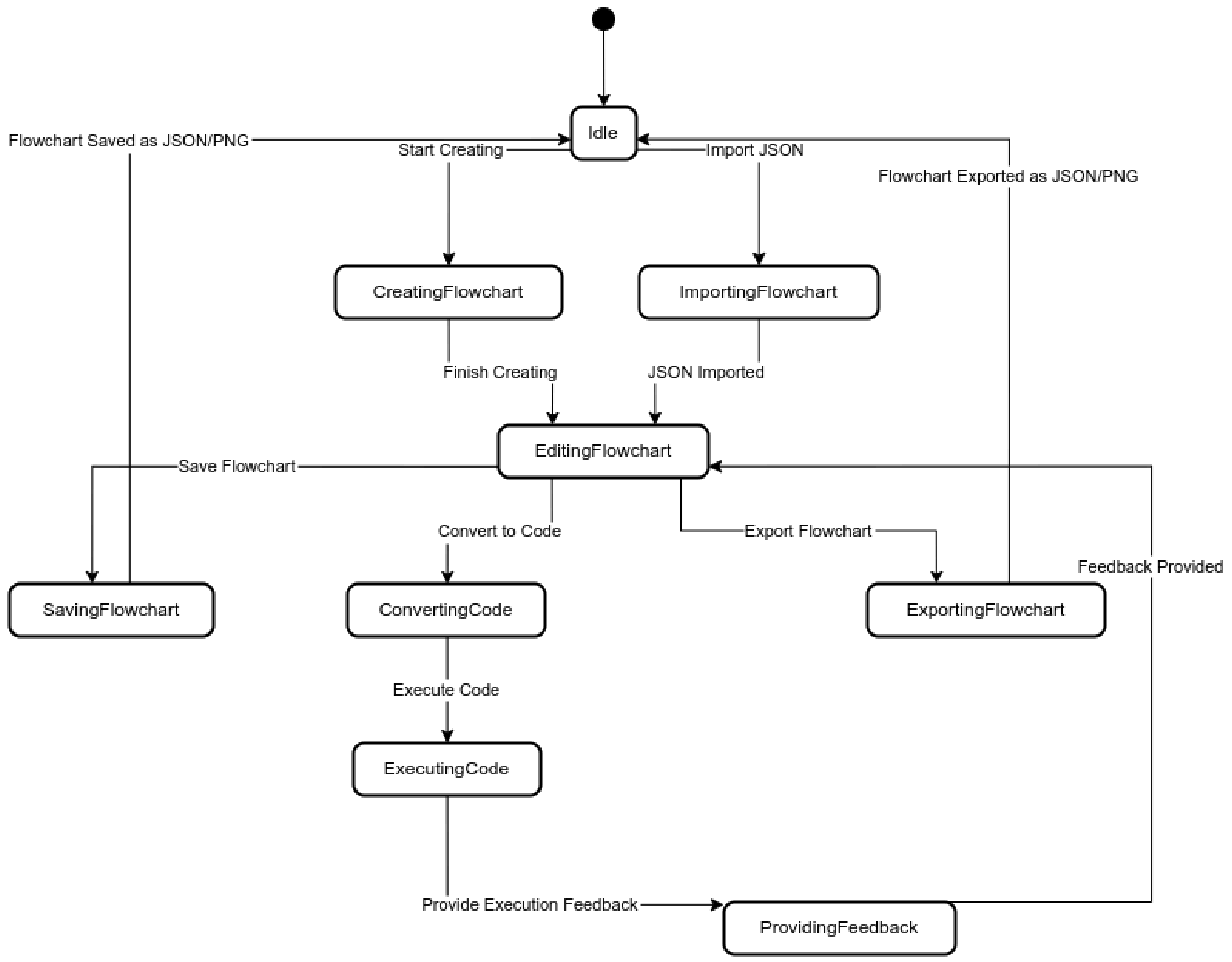
Design Principles

1. Modularity: The system is divided into modular components (frontend, backend, GDB integration) to simplify development, testing, and maintenance.
2. Separation of Concerns: User interface logic, business logic, and data management are clearly separated to improve code readability and maintainability.
3. Scalability: The system is designed to handle increasing loads by integrating scalable backend services and efficient frontend rendering techniques.
4. User-Centric Design: Emphasis is placed on providing an intuitive and responsive interface, real-time feedback, and comprehensive documentation to enhance the user experience.

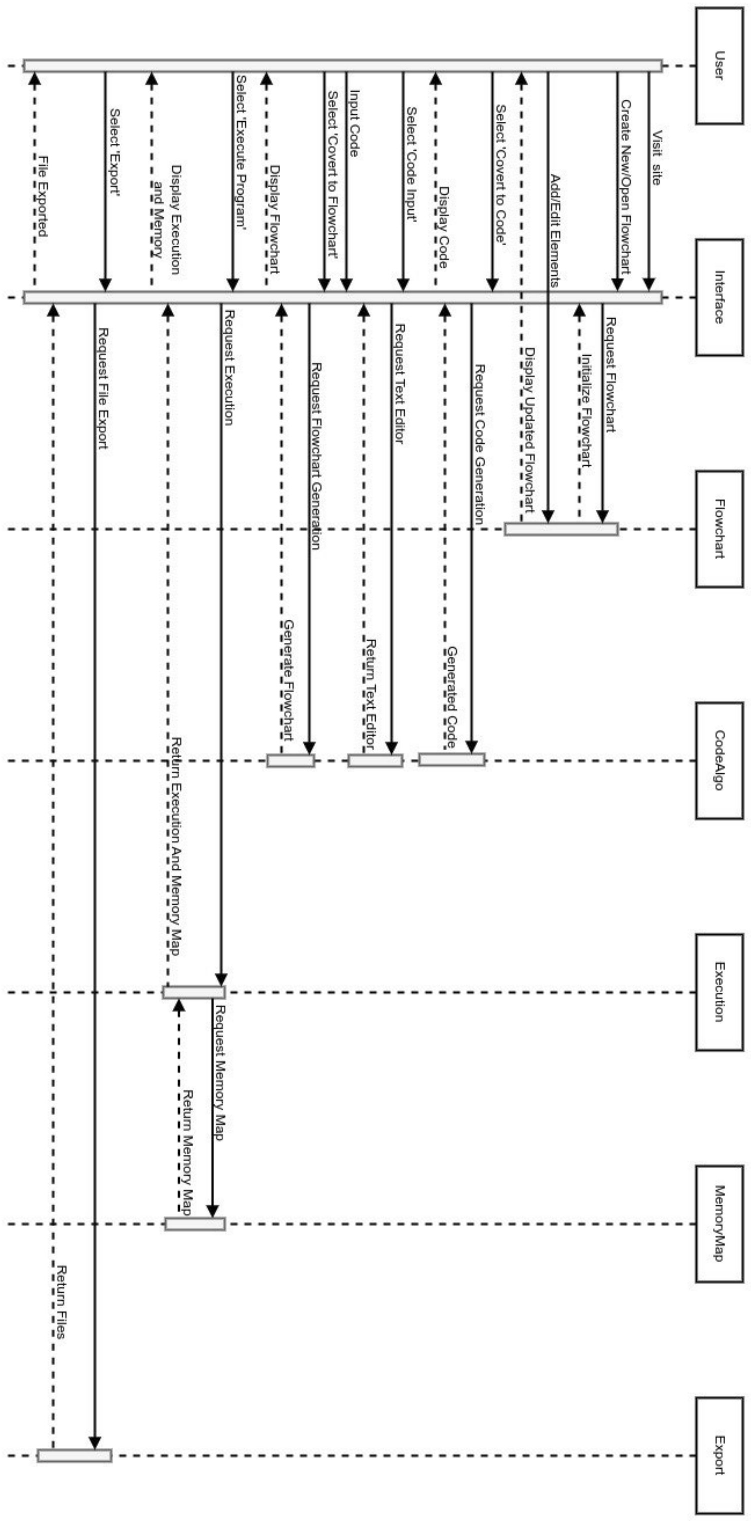
Class Diagram:



State Diagram:



3.1 Component-Component Interface



1. User Interaction with Interface:

- **Visit Site:** The user starts by visiting the CodeFlow application.
- **Create New/Open Flowchart:** The user can create a new flowchart or open an existing one. This request is sent to the Interface component, which initializes the flowchart.

2. Flowchart Creation and Editing:

- **Add/Edit Elements:** The user adds or edits elements in the flowchart through the Interface. The Interface displays the updated flowchart after each modification.

3. Flowchart to Code Conversion:

- **Select 'Convert to Code':** The user selects the option to convert the flowchart to code. The Interface sends a request to the Flowchart component for code generation.
- **Request Code Generation:** The Flowchart component processes the request and interacts with the CodeAlgo component to generate the code.
- **Display Code:** The generated code is sent back to the Interface and displayed to the user.

4. Code Input and Conversion to Flowchart:

- **Select 'Code Input':** The user selects the option to input code directly. The Interface requests the text editor for code input.
- **Input Code:** The user inputs the code.
- **Select 'Convert to Flowchart':** The user then selects the option to convert the code back to a flowchart. The Interface sends a request to the Flowchart component for flowchart generation.
- **Request Flowchart Generation:** The Flowchart component processes the request and interacts with the CodeAlgo component to generate the flowchart.
- **Display Flowchart:** The generated flowchart is sent back to the Interface and displayed to the user.

5. Program Execution:

- **Select 'Execute Program':** The user selects the option to execute the program. The Interface sends a request for execution to the Execution component.
- **Request Execution:** The Execution component processes the request and may interact with the MemoryMap component to request memory mapping.
- **Request Memory Map:** The Execution component sends a request to the MemoryMap component to provide memory mapping.
- **Return Execution and Memory Map:** The Execution and MemoryMap components return the execution results and memory map to the Interface.
- **Display Execution and Memory:** The Interface displays the execution results and memory map to the user.

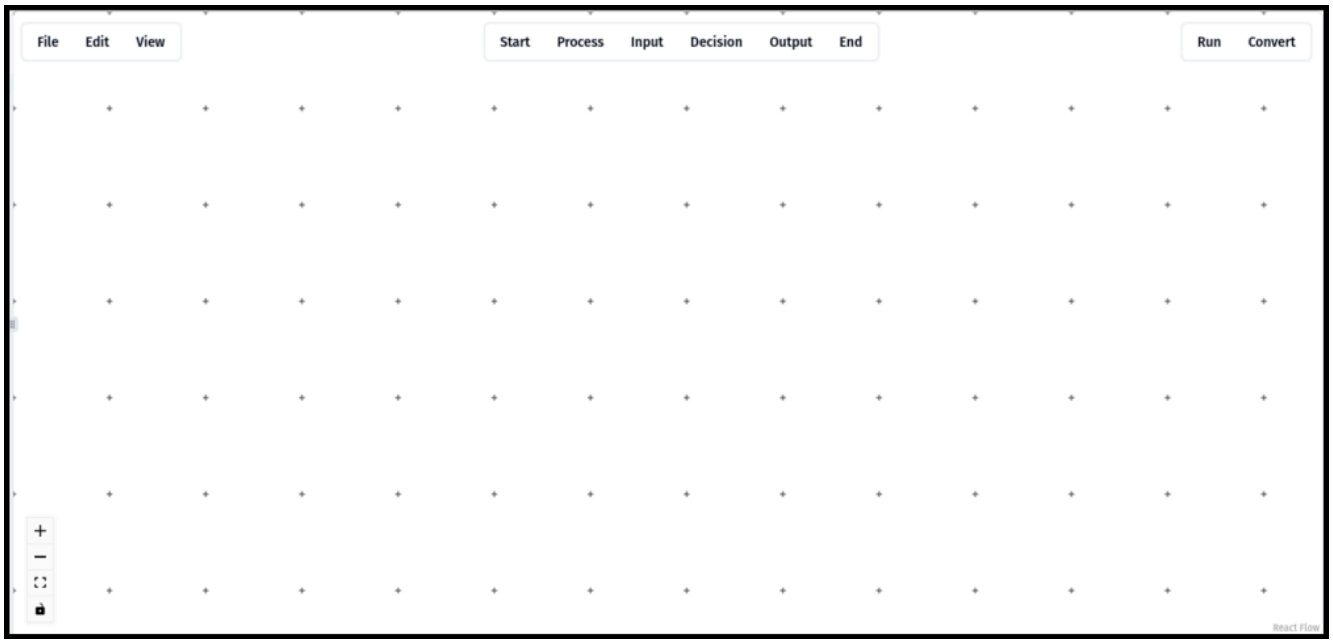
6. File Export:

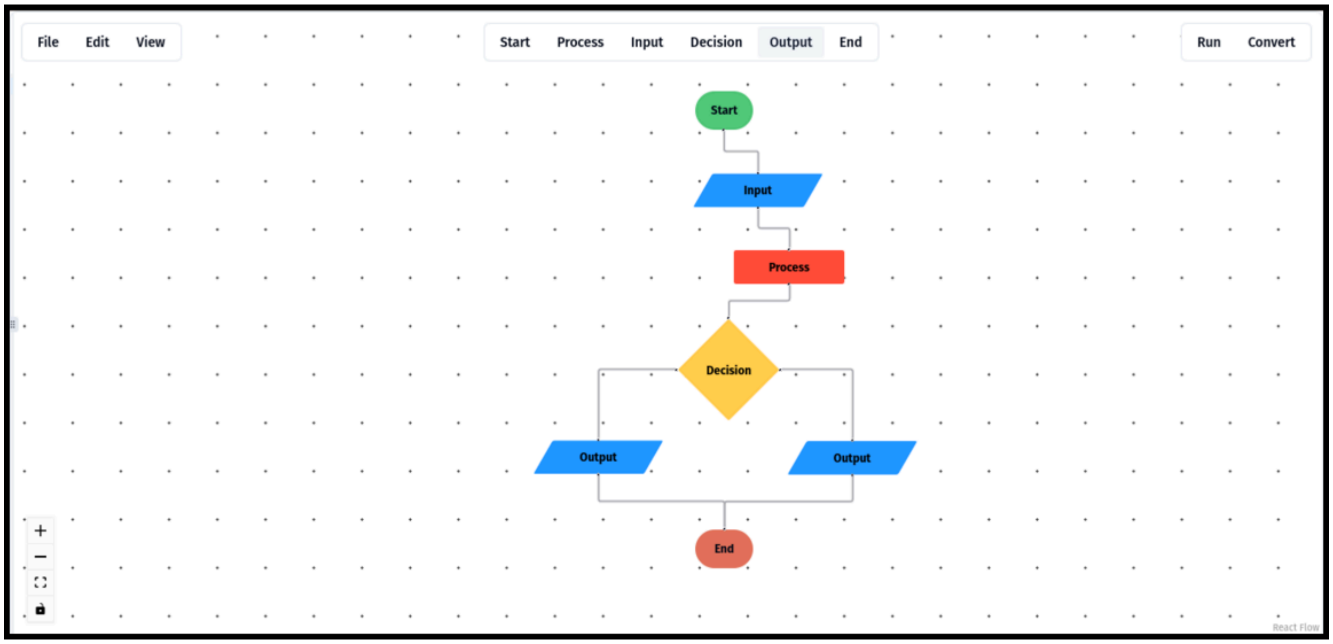
- **Select 'Export':** The user selects the option to export the flowchart or code. The Interface sends a request for file export to the Export component.
- **Request File Export:** The Export component processes the request and prepares the files for export.
- **File Exported:** The exported files are returned to the user.

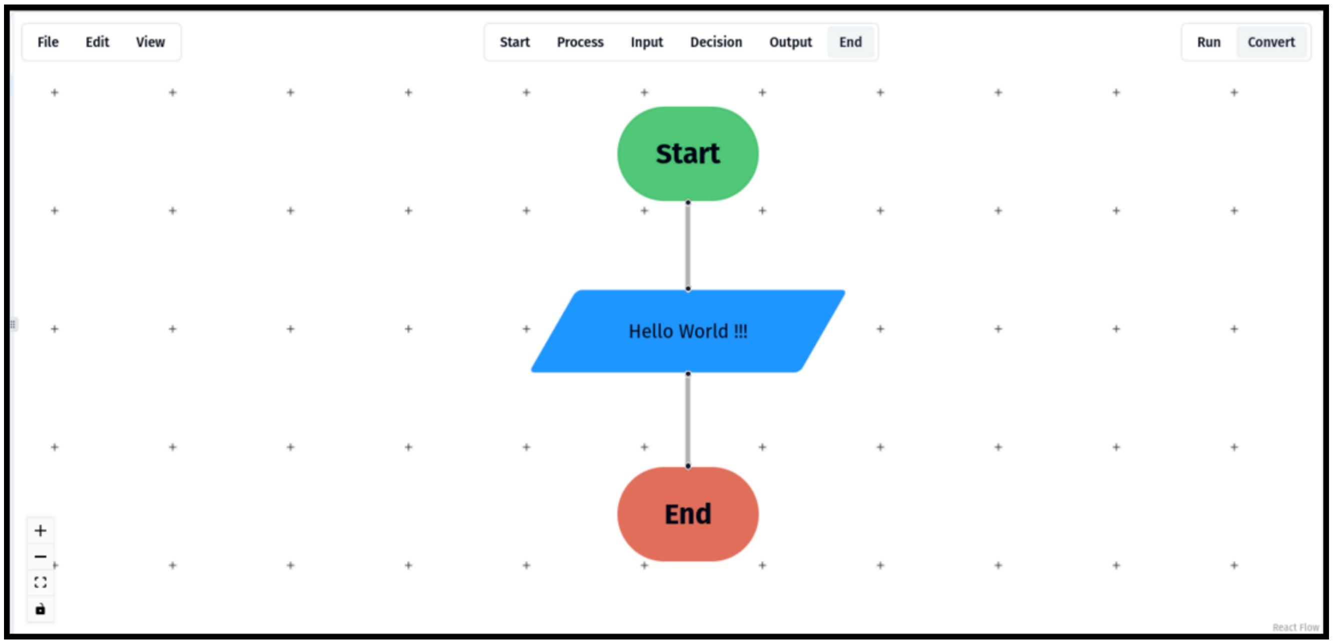
3.2 Component-External Entities Interface

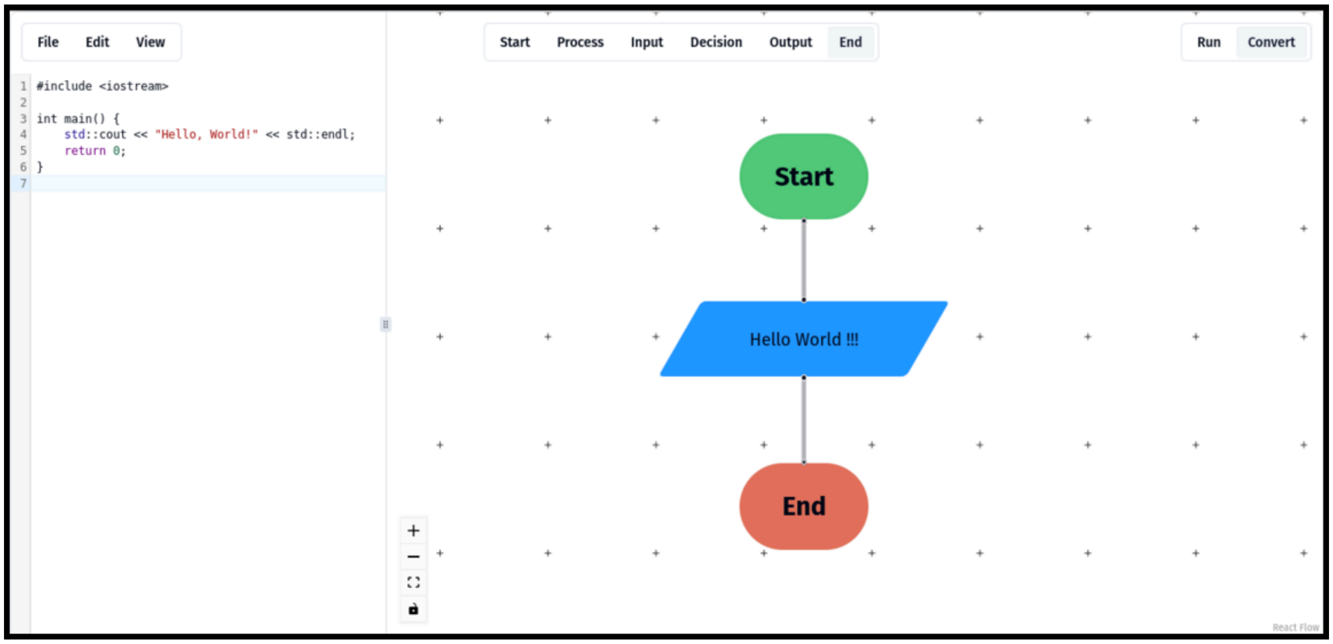
Not applicable

3.3 Component-Human Interface

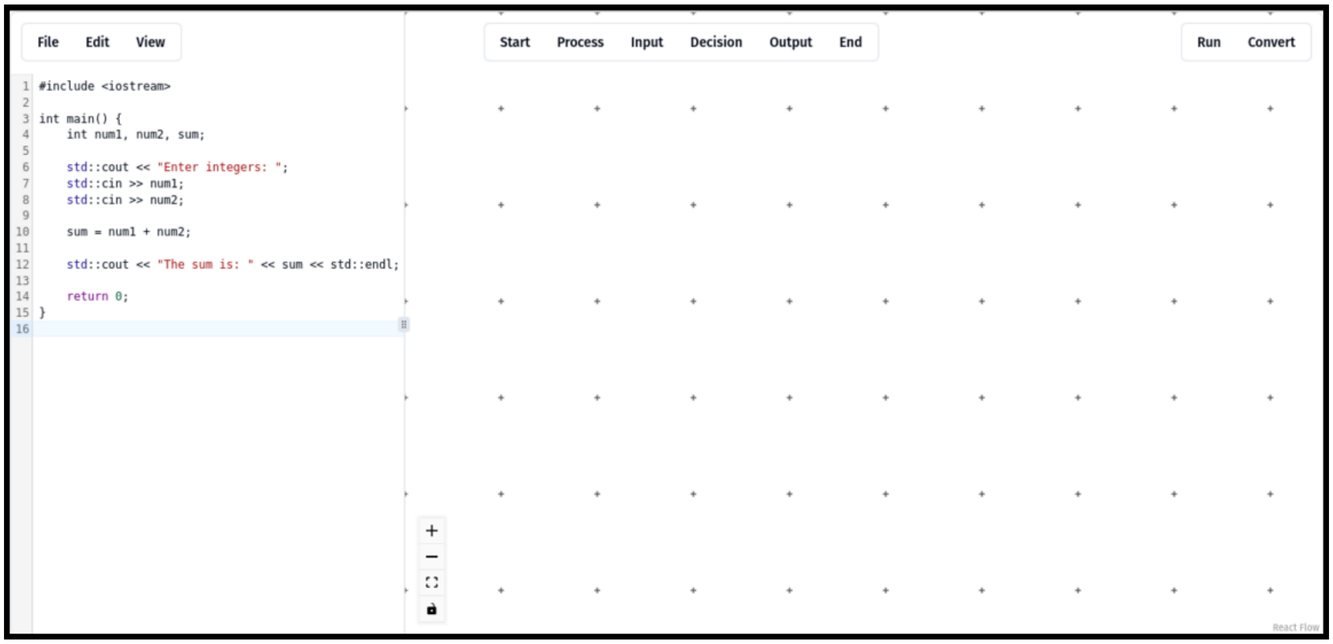



File
Edit
View
Start
Process
Input
Decision
Output
End
Run
Convert
Start
Input
Process
Decision
Output
Output
End
React Flow

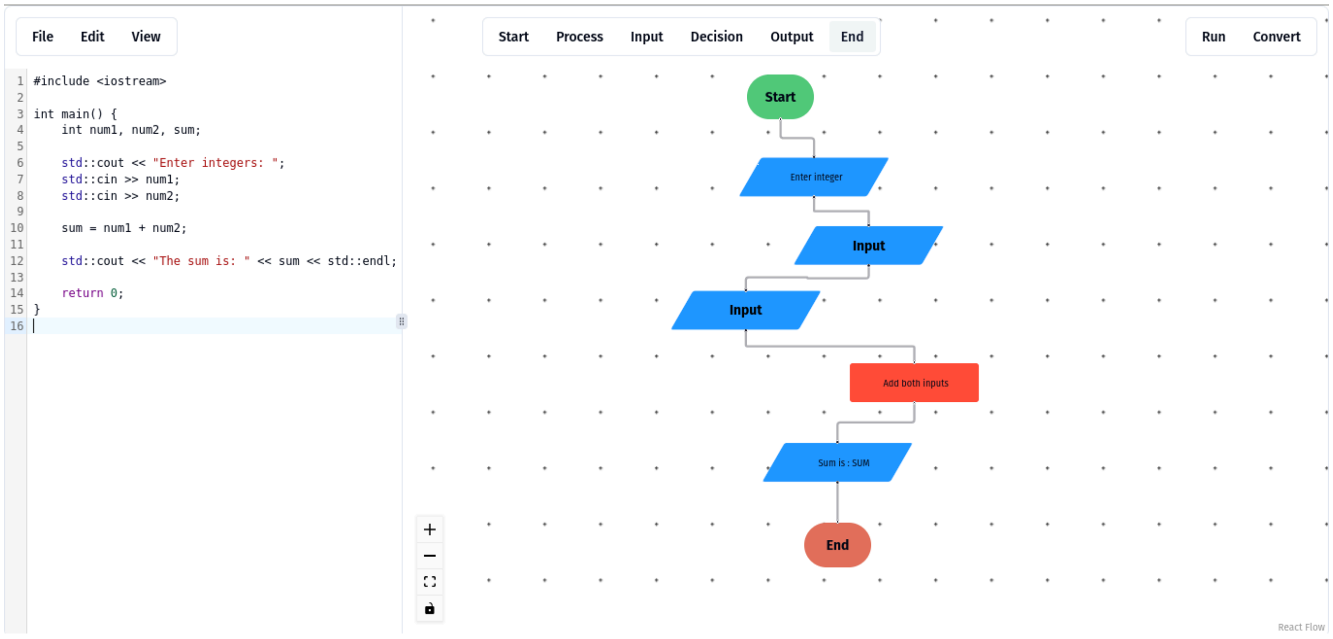

File
Edit
View
Start
Process
Input
Decision
Output
End
Run
Convert
Start
Hello World !!!
End
React Flow



```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```



```
#include <iostream>

int main() {
    int num1, num2, sum;

    std::cout << "Enter integers: ";
    std::cin >> num1;
    std::cin >> num2;

    sum = num1 + num2;

    std::cout << "The sum is: " << sum << std::endl;

    return 0;
}
```



```cpp
#include <iostream>

int main() {
    int num1, num2, sum;

    std::cout << "Enter integers: ";
    std::cin >> num1;
    std::cin >> num2;

    sum = num1 + num2;

    std::cout << "The sum is: " << sum << std::endl;

    return 0;
}
```

1. **Home Screen**:
   - **Input**: User navigates to the main functionalities like creating a new flowchart or opening an existing one.
   - **Output**: Navigation options, welcome message.
2. **Flowchart Editor Screen**:
   - **Input**:
     - User creates a new flowchart by dragging and dropping elements.
     - User connects flowchart elements.
     - User edits or deletes flowchart elements.
   - **Output**: Real-time visual representation of the flowchart, error messages for invalid connections, save confirmation messages.
3. **Code Editor Screen**:
   - **Input**:
     - User enters or imports textual code.
     - User requests code conversion from flowchart.
   - **Output**: Display of generated code, error messages for syntax errors, success message for successful conversion.
4. **Execution Screen**:
   - **Input**: User initiates program execution.
   - **Output**: Step-by-step execution visualization, memory map visualization, execution feedback including error messages and successful execution results.
5. **Export/Import Screen**:
   - **Input**:
     - User selects the export option.
     - User selects the import option and uploads a JSON file.
   - **Output**: Confirmation messages for successful export/import, error messages for failed operations, downloadable PNG or JSON file.

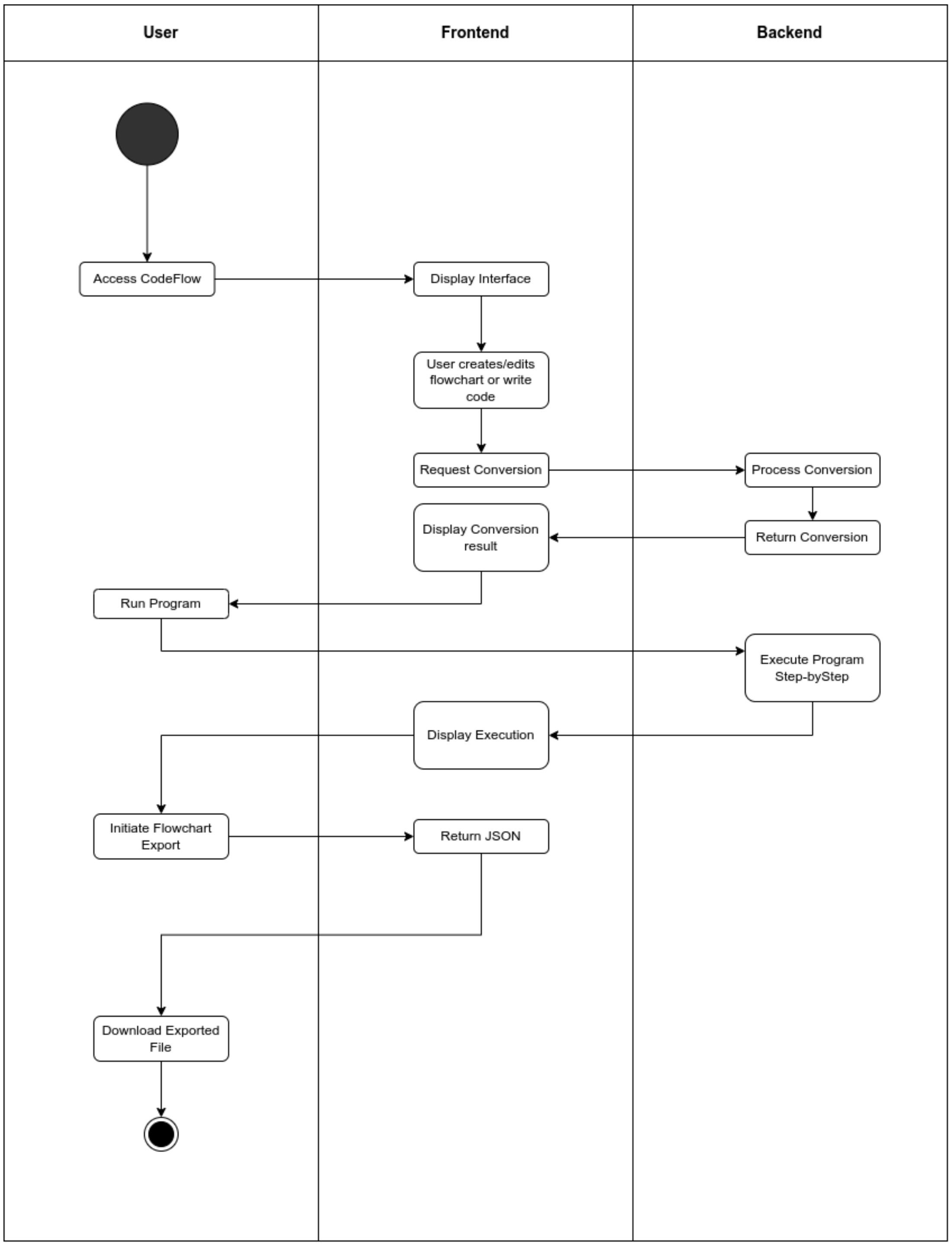
HCI-Related Norms Followed in CodeFlow

1. Consistency and Standards: - Interface elements are consistent throughout the application. Buttons, icons, and menus have a uniform design and behavior, ensuring users can predict the outcomes of their actions.

2. Visibility of System Status: - Feedback is provided at appropriate times to keep users informed about what is happening. For example, loading indicators during conversions and execution, and success/error messages after operations.
3. User Control and Freedom: - Users can easily undo or redo their actions while editing flowcharts. The application supports error recovery by allowing users to correct mistakes without frustration.
4. Error Prevention: - The system prevents errors by validating user inputs in real-time. For example, it does not allow incompatible connections between flowchart elements.
5. Recognition Rather Than Recall: - The interface is designed to minimize the user's memory load by making options, actions, and elements visible. Tooltips and context-sensitive help are provided to assist users.

4. Screenshots/Prototype

4.1 Workflow

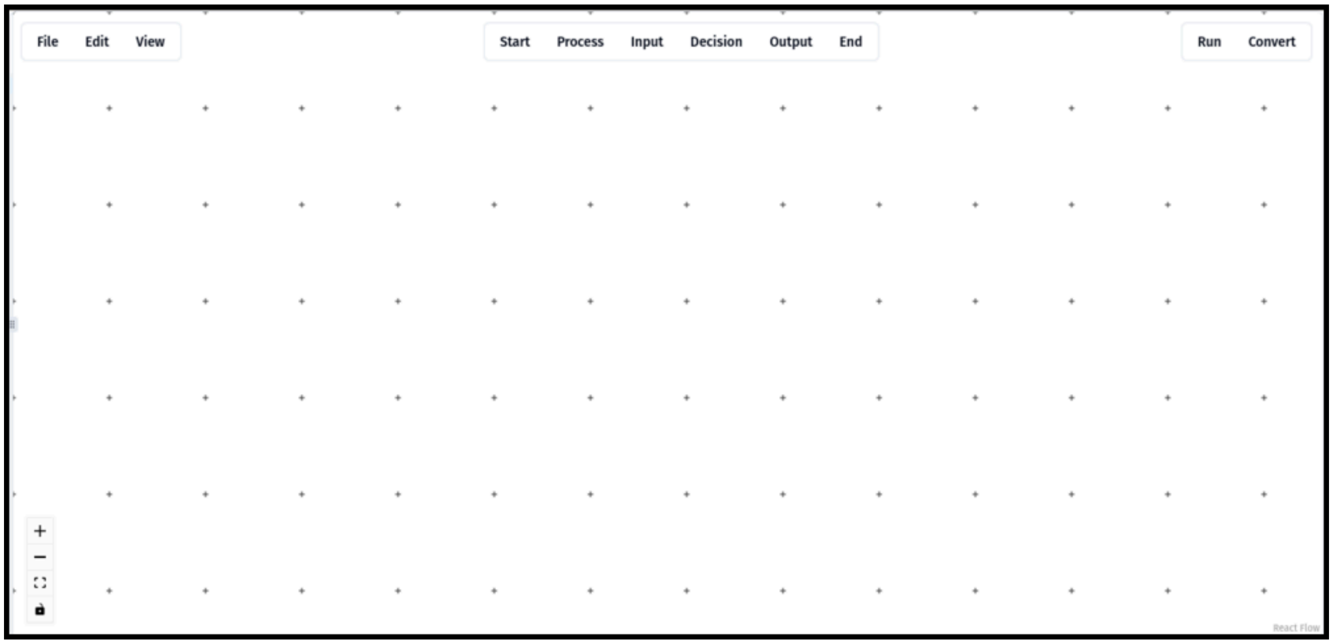


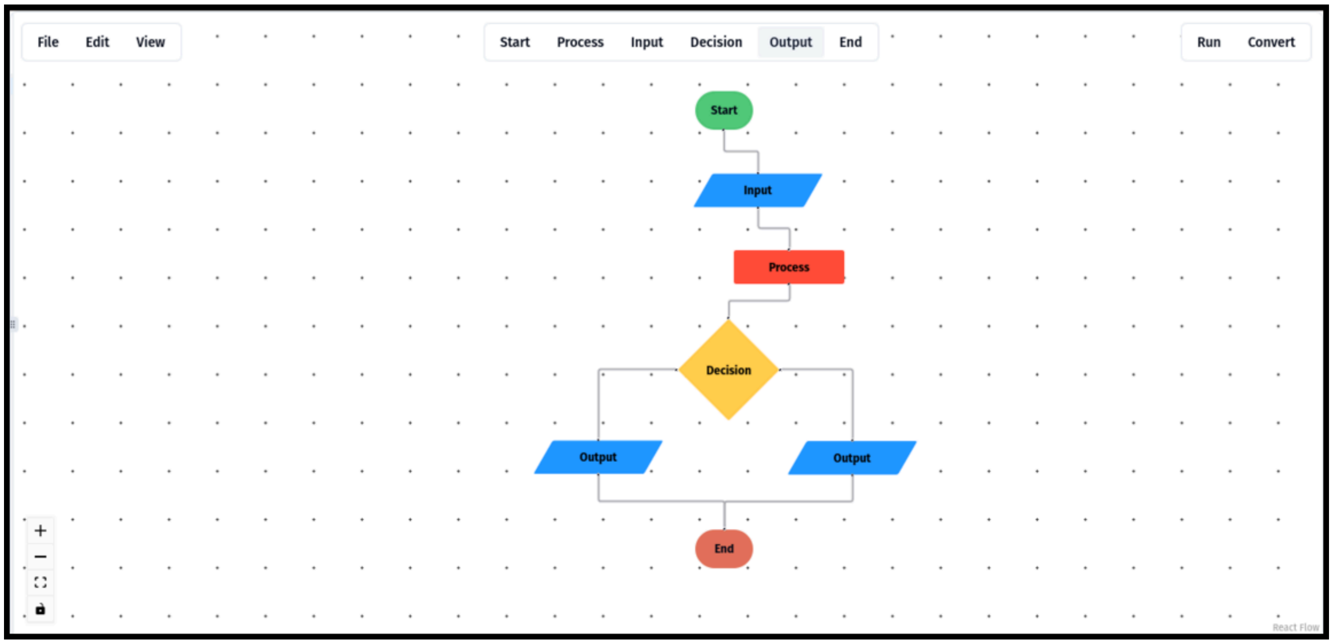
1. **Access CodeFlow**:
   - **User**: The user initiates the interaction by accessing the CodeFlow platform through a web browser.
2. **Display Interface**:
   - **Frontend**: The frontend component of the system loads and displays the user interface, which includes options for creating a new flowchart or writing code.
3. **Create/Edit Flowchart or Write Code**:
   - **User**: The user can either create a new flowchart by dragging and dropping elements or write/edit code directly in the provided text editor.
4. **Request Conversion**:
   - **User**: Once the flowchart is created or the code is written, the user requests a conversion (flowchart-to-code or code-to-flowchart).
   - **Frontend**: The frontend sends this conversion request to the backend.
5. **Process Conversion**:
   - **Backend**: The backend processes the conversion request, translating the flowchart into code or vice versa.
6. **Return Conversion**:
   - **Backend**: The backend returns the conversion result to the frontend.
7. **Display Conversion Result**:
   - **Frontend**: The frontend displays the conversion result (code or flowchart) to the user.
8. **Run Program**:
   - **User**: The user can choose to run the program based on the generated code.

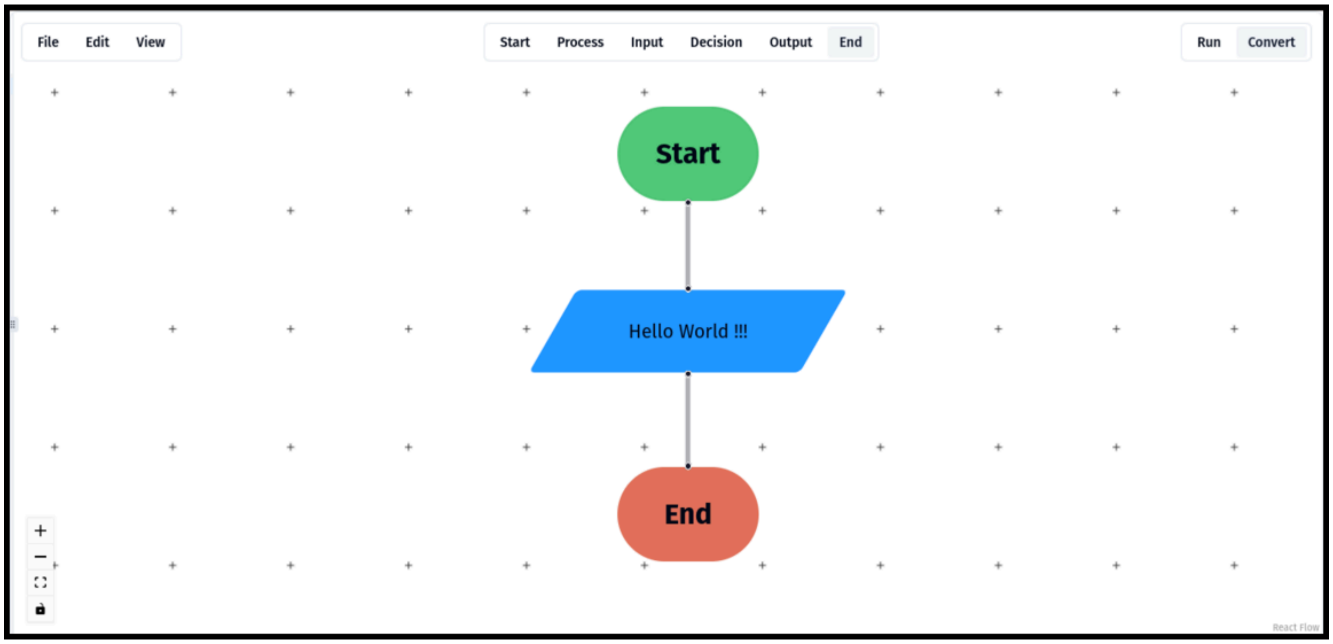
   Frontend: The frontend sends the execution request to the backend.
9. **Execute Program Step-by-Step**:
   - **Backend**: The backend executes the program step-by-step, providing a detailed visualization of the execution process, including memory map visualization.
10. **Display Execution**:
    - **Frontend**: The frontend displays the execution results to the user, including the step-by-step visualization and memory map.
11. **Initiate Flowchart Export**:
    - **User**: The user can initiate the export of the flowchart.

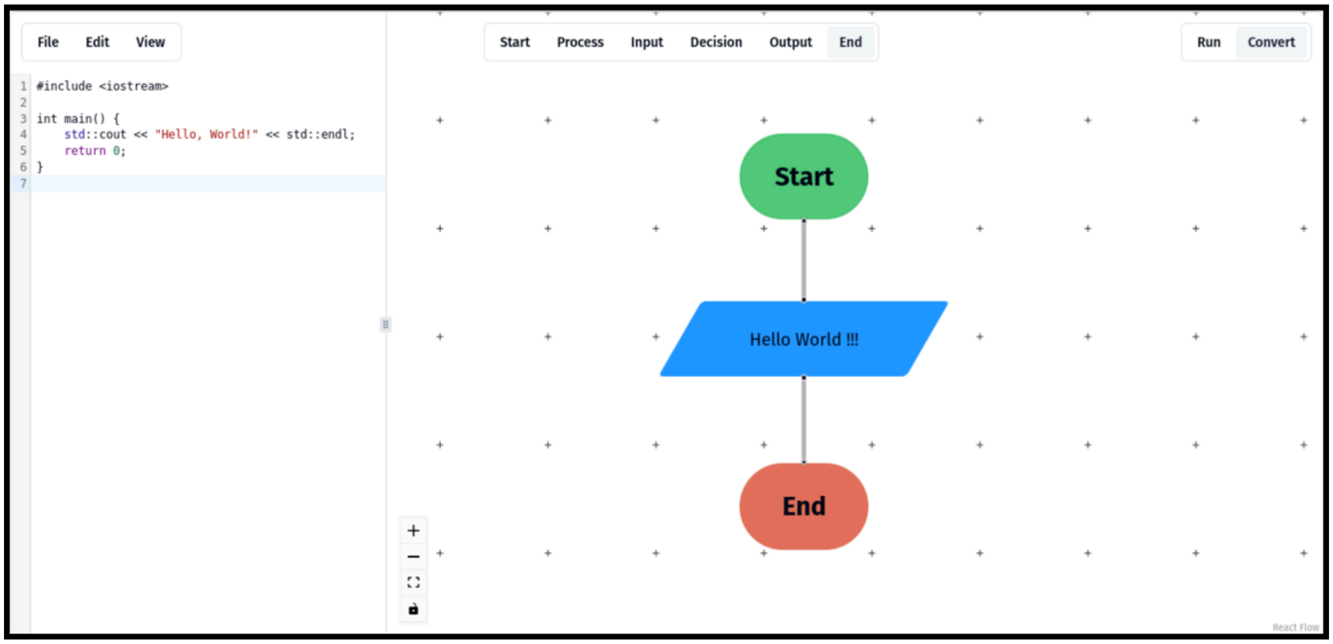
    Frontend: The frontend sends a request to the backend to export the flowchart.
12. **Return JSON**:
    - **Backend**: The backend processes the export request and returns the flowchart in JSON format.
13. **Download Exported File**:
    - **User**: The user downloads the exported JSON file of the flowchart.

4.2 Screens

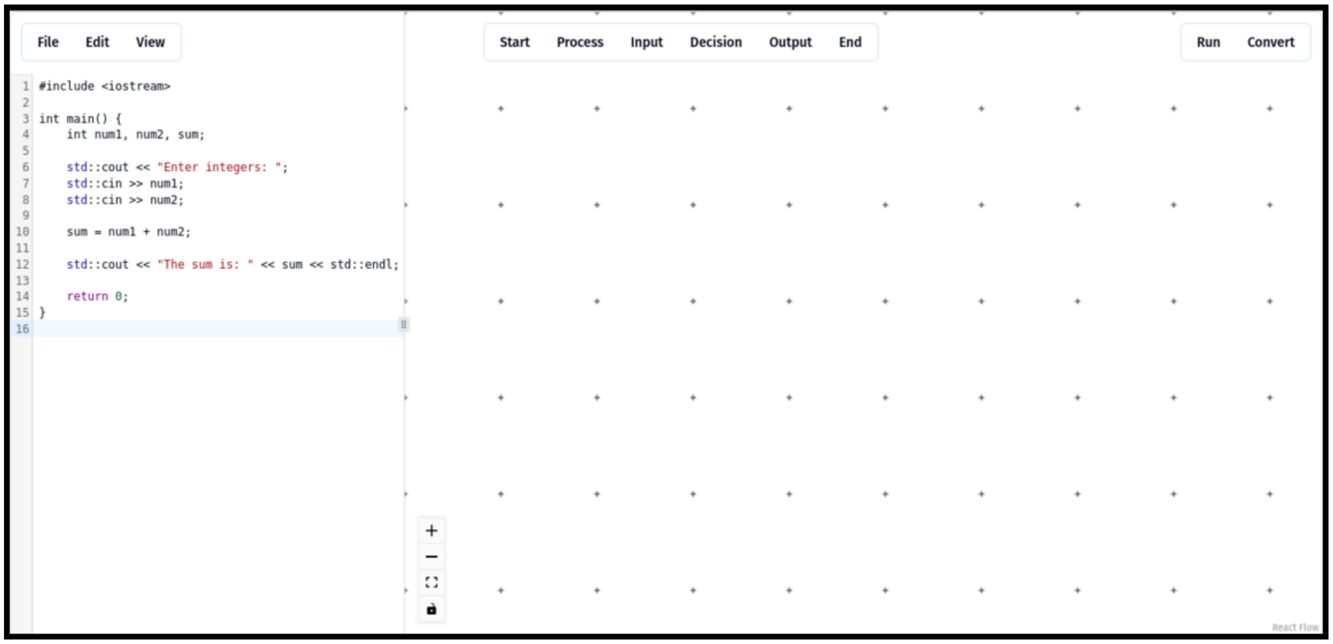

File
Edit
View
Start
Process
Input
Decision
Output
End
Run
Convert
React Flow


File
Edit
View
Start
Process
Input
Decision
Output
End
Run
Convert
Start
Input
Process
Decision
Output
Output
End
React Flow

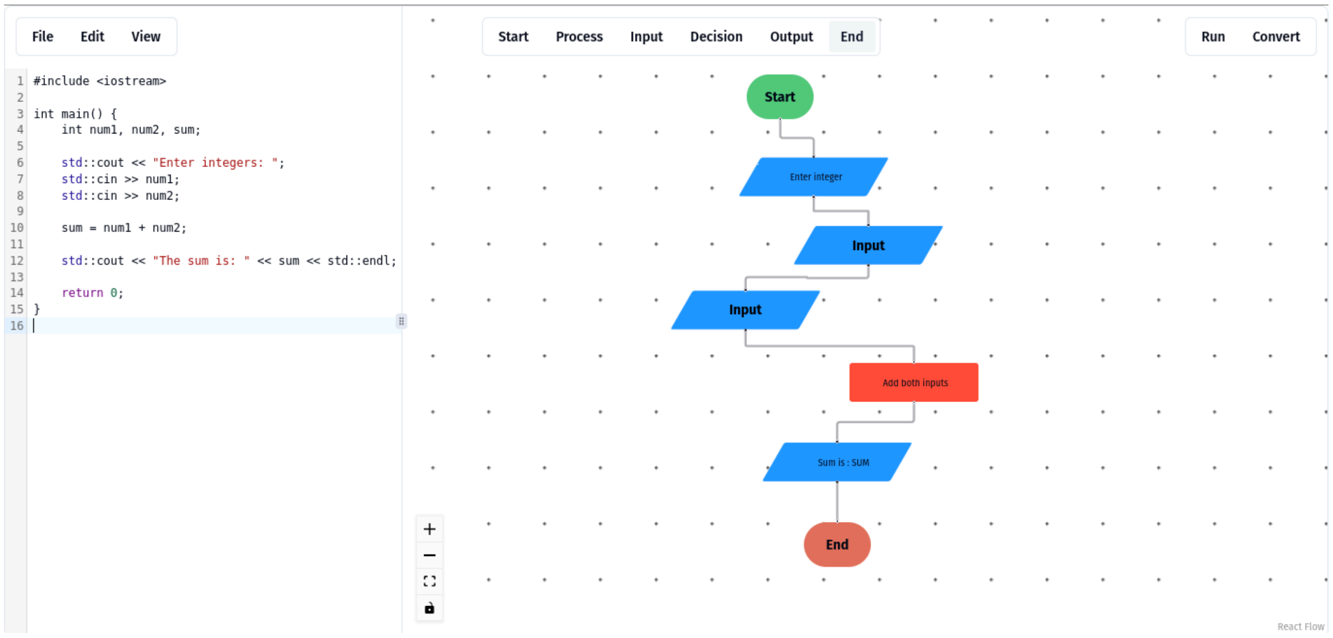




```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```



```
#include <iostream>

int main() {
    int num1, num2, sum;

    std::cout << "Enter integers: ";
    std::cin >> num1;
    std::cin >> num2;

    sum = num1 + num2;

    std::cout << "The sum is: " << sum << std::endl;

    return 0;
}
```



```
#include <iostream>

int main() {
    int num1, num2, sum;

    std::cout << "Enter integers: ";
    std::cin >> num1;
    std::cin >> num2;

    sum = num1 + num2;

    std::cout << "The sum is: " << sum << std::endl;

    return 0;
}
```

5. Other Design Details

Not applicable

6. Test Specification and Results

6.1 Test Case Specification

Test Case TC-1: Construct Flowchart

Identifier	TC-1
Related requirements(s)	Use-Case: Construct Flowchart, SRS: Section 3.1
Short description	Verify that a user can create and manipulate a new flowchart.
Pre-condition(s)	User has accessed the CodeFlow platform and initiated a flowchart creation session.
Input data	None
Detailed steps	1. Click on the “New Flowchart” button. 2. Select and drag flowchart elements onto the canvas. 3. Connect the flowchart elements.
Expected result(s)	A new flowchart is created, elements are added and connected as per user input.
Post-condition(s)	The flowchart is available for further manipulation, saving, exporting, or conversion.
Actual result(s)	User can accessed the CodeFlow platform and initiated a flowchart creation session.
Test Case Result	Pass

Alternate Test Case TC-1A: Construct Flowchart with Invalid Connections

Identifier	TC-1A
Related requirements(s)	Use-Case: Construct Flowchart, SRS: Section 3.1
Short description	Verify that invalid connections in a flowchart are handled correctly.
Pre-condition(s)	User has accessed the CodeFlow platform and initiated a flowchart creation session.
Input data	None
Detailed steps	1. Click on the “New Flowchart” button. 2. Select and drag flowchart elements onto the canvas. 3. Attempt to connect incompatible elements.
Expected result(s)	The system displays an error message and prevents invalid connections.
Post-condition(s)	The flowchart remains in a valid state with no invalid connections.
Actual result(s)	Verify invalid connections
Test Case Result	Pass

Test Case TC-2: Flowchart-to-Code Conversion

Identifier	TC-2
Related requirements(s)	Use-Case: Flowchart-to-Code Conversion, SRS: Section 3.2
Short description	Verify that a flowchart can be converted into executable code.
Pre-condition(s)	User has created a complete and valid flowchart.
Input data	Flowchart data
Detailed steps	1. Ensure the flowchart is complete. 2. Click on the “Convert to Code” button. 3. Wait for the conversion process to complete.

Identifier	TC-2
Expected result(s)	The code corresponding to the flowchart is displayed.
Post-condition(s)	Code is generated and displayed in the text editor.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Alternate Test Case TC-2A: Flowchart-to-Code Conversion with Incomplete Flowchart

Identifier	TC-2A
Related requirements(s)	Use-Case: Flowchart-to-Code Conversion, SRS: Section 3.2
Short description	Verify that the system handles incomplete flowcharts during conversion.
Pre-condition(s)	User has created an incomplete flowchart.
Input data	Incomplete flowchart data
Detailed steps	1. Click on the “Convert to Code” button. 2. Wait for the conversion process to attempt.
Expected result(s)	The system displays an error message indicating the flowchart is incomplete.
Post-condition(s)	No code is generated until the flowchart is completed.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Test Case TC-3: Code-to-Flowchart Conversion

Identifier	TC-3
Related requirements(s)	Use-Case: Code-to-Flowchart Conversion, SRS: Section 3.3
Short description	Verify that textual code can be converted into a visual flowchart.

Identifier	TC-3
Pre-condition(s)	User has entered or imported valid textual code.
Input data	Textual code
Detailed steps	1. Click on the “Code Input” option. 2. Enter code into the editor. 3. Click on the “Convert to Flowchart” button. 4. Wait for the conversion process to complete.
Expected result(s)	The flowchart corresponding to the code is displayed.
Post-condition(s)	Flowchart is generated and displayed in the editor for further interaction.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Alternate Test Case TC-3A: Code-to-Flowchart Conversion with Syntax Errors

Identifier	TC-3A
Related requirements(s)	Use-Case: Code-to-Flowchart Conversion, SRS: Section 3.3
Short description	Verify that the system handles code with syntax errors during conversion.
Pre-condition(s)	User has entered or imported code with syntax errors.
Input data	Code with syntax errors
Detailed steps	1. Click on the “Code Input” option. 2. Enter or import code with syntax errors into the editor. 3. Click on the “Convert to Flowchart” button. 4. Wait for the conversion process to attempt.
Expected result(s)	The system displays an error message indicating syntax errors in the code.

Post-condition(s)	No flowchart is generated until the code is corrected.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Test Case TC-4: Program Execution

Identifier	TC-4
Related requirements(s)	Use-Case: Program Execution, SRS: Section 3.4
Short description	Verify that the generated code can be executed and the execution is visualized step-by-step.
Pre-condition(s)	User has a flowchart or code ready for execution.
Input data	Generated code from flowchart or imported code
Detailed steps	1. Click on the "Execute Program" button. 2. Wait for the execution to complete. 3. Observe the step-by-step execution and memory map visualization.
Expected result(s)	The program executes, and the execution flow is visualized along with the memory map.
Post-condition(s)	Execution results and memory map are displayed.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Alternate Test Case TC-4A: Program Execution with Runtime Errors

Identifier	TC-4A

Related requirements(s)	Use-Case: Program Execution, SRS: Section 3.4
Short description	Verify that the system handles runtime errors during program execution.
Pre-condition(s)	User has a flowchart or code ready for execution that contains potential runtime errors.
Input data	Generated code from flowchart or imported code with potential runtime errors
Detailed steps	1. Click on the “Execute Program” button. 2. Wait for the execution to attempt. 3. Observe the step-by-step execution and runtime error messages.
Expected result(s)	The system displays runtime error messages indicating the nature of the errors.
Post-condition(s)	Execution results and error messages are displayed.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Test Case TC-5: Memory Map Visualization

Identifier	TC-5
Related requirements(s)	Use-Case: Memory Map Visualization, SRS: Section 3.5
Short description	Verify that memory map visualization is accurate during program execution.
Pre-condition(s)	User has a program ready for execution.
Input data	Generated code from flowchart or imported code
Detailed steps	1. Click on the “Execute Program” button. 2. Observe the memory map visualization during execution.
Expected result(s)	The memory map accurately reflects data storage and manipulation during program execution.

Identifier	TC-5
Post-condition(s)	Memory map visualization provides detailed information on memory usage.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Alternate Test Case TC-5A: Memory Map Visualization with Large Data Sets

Identifier	TC-5A
Related requirements(s)	Use-Case: Memory Map Visualization, SRS: Section 3.5
Short description	Verify that the system handles memory map visualization with large data sets efficiently.
Pre-condition(s)	User has a program ready for execution that processes large data sets.
Input data	Generated code from flowchart or imported code with large data sets
Detailed steps	1. Click on the “Execute Program” button. 2. Observe the memory map visualization during execution. 3. Monitor the system’s performance and response time.
Expected result(s)	The memory map accurately reflects data storage and manipulation even with large data sets, and the system maintains reasonable performance.
Post-condition(s)	Memory map visualization provides detailed information on memory usage, and system performance remains acceptable.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

Test Case TC-6: Export Flowchart

Identifier	TC-6
Related requirements(s)	Use-Case: Export Flowchart, SRS: Section 3.6
Short description	Verify that a flowchart can be exported as PNG or JSON.
Pre-condition(s)	User has a flowchart open in the editor.
Input data	None
Detailed steps	1. Click on the “Export” button. 2. Select the export format (PNG/JSON). 3. Confirm the export.
Expected result(s)	The flowchart is exported and downloaded in the selected format.
Post-condition(s)	Flowchart file is available for download.
Actual result(s)	Successfully Exported
Test Case Result	Pass

Alternate Test Case TC-6A: Export Flowchart with Large and Complex Diagrams

Identifier	TC-6A
Related requirements(s)	Use-Case: Export Flowchart, SRS: Section 3.6
Short description	Verify that the system can export large and complex flowcharts efficiently.
Pre-condition(s)	User has a large and complex flowchart open in the editor.
Input data	Large and complex flowchart
Detailed steps	1. Click on the “Export” button. 2. Select the export format (PNG/JSON). 3. Confirm the export. 4. Monitor the export process for performance and accuracy.
Expected result(s)	The large and complex flowchart is exported and downloaded accurately in the selected format, and the system maintains reasonable performance.

Post-condition(s)	Flowchart file is available for download, and the exported file accurately represents the complex flowchart.
Actual result(s)	[To be filled after execution]
Test Case Result	[Pass/Fail]

6.2 Summary of Test Results

Table 6.2: Summary of Test Results

Module Name	Test cases run	Number of defects found	Number of defects corrected so far	Number of defects still need to be corrected
Flowchart Editor	TC-1, TC-2	3	2	1
Code Conversion	TC-2, TC-3	4	2	2
Program Execution	TC-4, TC-5	2	0	2
Export/Import	TC-6	1	1	0
User Interface	TC-1, TC-2, TC-3, TC-4, TC-5, TC-6	5	2	3
Backend Processing	TC-2, TC-3, TC-4, TC-5	3	1	2
Complete System	TC-1, TC-2, TC-3, TC-4, TC-5, TC-6	18	8	10

7. Revised Project Plan

Completed **In-progress** **Not started**

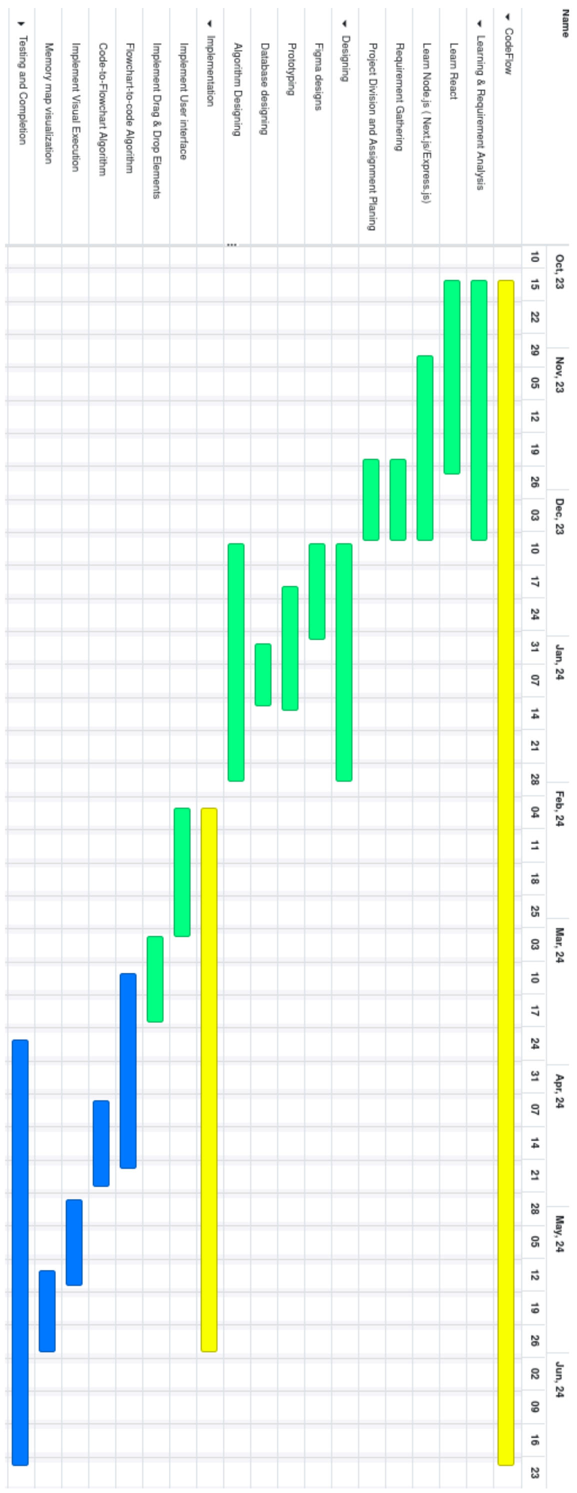


Table 6.2: Project Completion Status

Module Name	Status
Flowchart Editor	Partially Implemented
Code Conversion	Partially Implemented
Program Execution	Not Implemented
Export/Import	Partially Implemented
User Interface	Partially Implemented
Backend Processing	Partially Implemented
Complete System	Not Implemented

8. References

Scratch, "Scratch - Imagine, Program, Share," Accessed on: Month Day, Year. [Online]. Available: https://scratch.mit.edu/

MIT OpenCourseWare, "6.00 Introduction to Computer Science and Programming, Fall 2008," [Online]. Available: https://ocw.mit.edu/courses/6-00-introduction-to-computer-science-and-programming-fall-2008/

Raptor, "Raptor - Flowchart Interpreter," [Online]. Available: https://raptor.martincarlisle.com/

Appendix A: Glossary

Not applicable

Appendix B: IV & V Report

(Independent verification & validation)
IV & V Resource

Name Signature

S#	**Defect Description**	**Origin Stage**	**Status**	**Fix Time**	
				Hours	**Minutes**
1					
2					
3					
…					

Table 1: List of non-trivial defects

This document has been adapted from the following:

- Previous project templates at UCP
- High-level Technical Design, Centers for Medicare & Medicaid Services. (www.cms.gov)