School of Computing

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Interactive Learning: An Android Application for Early Computer Science Education to Explore and Learn Algorithms

Abderrahmane Bennabet

Submitted in accordance with the requirements for the degree of Computer science.

2023/2024

COMP3931 Individual Project

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (01/05/24)*
Link to the video demonstration	*URL*	*Uploaded to Minerva (25/04/24)*
Link to the online code repository	*URL*	*Sent to supervisor and assessor (25/04/24)*

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)

Abstract

This paper explores the development of an Android application designed to enhance the learning experience of early computer science students by providing an interactive platform for understanding algorithms. The application includes a suite of three fundamental algorithms, each detailed in design, implementation, and user interaction. By leveraging intuitive user interfaces, the application allows students to actively engage with each algorithm through real-time visualizations and step-by-step execution, reinforcing theoretical concepts with practical application. The development process employed agile methodologies to ensure adaptability and responsiveness to user feedback, facilitating a user-centered design approach. Additionally, the application incorporates assessment tools to evaluate student progress and understanding, fostering an educational environment that is both informative and engaging. This paper details the technical architecture and educational theory underpinning the application. It discusses the potential impact on pedagogical practices in computer science education, emphasizing the role of interactive learning tools in facilitating a deeper understanding of complex algorithms.

Acknowledgments

I would like to express my profound gratitude to my supervisor, Arshad Jhumka, for his consistent and invaluable support throughout this project. My gratitude extends to my assessor, Jie Xu, for his guidance and assistance.

Special thanks are due to my parents, Hakim and Meriem, and my siblings, Anfel and Ritel, for their enduring love and encouragement throughout this journey. Their unwavering support has been a cornerstone of my success.

Table of Contents

Chapter 1: Introduction

1.1 Introduction

In the rapidly evolving field of computer science, understanding the fundamentals of data structures and algorithms is essential for students and professionals. These foundational concepts are crucial for academic success and play a pivotal role in solving real-world problems efficiently. However, the abstract nature of these topics often poses significant learning challenges. Traditional methods, while valuable, sometimes fall short of illustrating the dynamic behavior of algorithms and the adaptability of data structures in a manner that resonates with all learners.

The project recognizes this educational gap and introduces an interactive learning platform to demystify algorithms and data structures through visualization and hands-on interaction. By allowing users to manipulate data and observe the real-time execution of algorithms, the platform aims to foster a deeper understanding and retention of these critical computer science concepts.

1.2 Project Overview

This report details the development of an interactive algorithm visualization application, "**AlgoLearn**," from its inception to its final state. The project was conceived to enhance computer science education by providing a more engaging and effective way for students to learn about algorithms and data structures. To achieve this, AlgoLearn leverages modern software development techniques and educational theories to create a user-friendly environment where learners can explore and experiment with various algorithms at their own pace.

The application has diverse algorithms, emphasizing Dijkstra's algorithm for finding the shortest paths in a graph, the insertion sort algorithm for array sorting, and the depth-first search (DFS) algorithm for traversing or searching graph data structures. These algorithms are selected for their distinct educational value and capacity to illustrate fundamental algorithm design and analysis concepts across different data structures. Dijkstra's and DFS algorithms teach learners how to navigate through graphs or find the shortest path, and insertion sort provides a hands-on understanding of basic sorting mechanisms and array manipulation. The application aims to provide a comprehensive learning experience that bridges the gap between abstract algorithmic theories and their practical applications in real-world problem-solving by covering graph and array-based algorithms.

Throughout the development process, we prioritized flexibility by enabling users to customize graphs and datasets. This allows users to explore various scenarios and outcomes and shape their learning journey. Users can edit their graphs at any stage and save them for future reference, enhancing a collaborative and tailored learning experience.

The rest of the report is structured as follows: Chapter 2 offers a comprehensive background study, highlighting the significance of algorithms and data structures through a detailed literature review and examining existing solutions and their limitations. Chapter 3 describes the selected methodologies and technologies that shaped the project with the project design. Chapter 4 provides an in-depth look into the implementation details, including user interaction aspects of algorithm visualizations. Chapter 5 covers evaluation, which includes testing and validation. Lastly, Chapter 6 summarizes the project's outcomes, addresses the challenges encountered, and suggests avenues for future enhancements.

Chapter 2: Background Research

2.1 Importance of Data Structures and Algorithms

Algorithms in computing are precise sets of instructions for solving specific problems or performing tasks. They dictate a sequence of steps that, when followed, yield a solution in a finite amount of time. Similarly, data structures are systematic ways of organizing, managing, and storing data so that it can be accessed and modified efficiently. Together, algorithms and data structures form the core of computer science, equipping developers with the tools to tackle complex problems by breaking them down into manageable, logical sequences and organizing data in ways that optimize performance and resource utilization.

The utility of data structures lies in their ability to organize, store, and manage data effectively. Bhargava (2016) posits that mastering data structures equips programmers with the tools to implement dynamic and scalable algorithms that are crucial for modern software applications. Whether navigating through extensive databases with graphs or leveraging arrays for data manipulation, the strategic choice of data structures can influence the functionality and efficiency of programs.

The essence of algorithms in computing extends to defining clear, executable steps for solving problems or performing tasks. Goodrich, Tamassia, and Goldwasser (2014) emphasize that the algorithmic approach to problem-solving is integral to developing effective, optimally efficient, and scalable software solutions. From everyday applications in search engines and social media platforms to specialized uses in scientific computing and artificial intelligence, algorithms facilitate the processing and analysis of data on an unprecedented scale.

Beyond their technical applications, the study of data structures and algorithms enriches the educational journey of computer science students. According to Sahami et al. (2013), integrating these subjects early in computer science curricula fosters analytical thinking and problem-solving skills, preparing students for advanced theoretical and practical challenges. Interactive learning tools and visualizations have emerged as transformative resources, offering an immersive experience that aids in conceptualizing and applying these abstract concepts more effectively (Porter et al., 2013).

2.2 Traditional Teaching Methods

The pedagogical landscape for teaching data structures and algorithms has traditionally been anchored in lectures, textbooks, and hands-on programming exercises. These conventional methods, while foundational, often face challenges in fully engaging students and facilitating a comprehensive understanding of these complex subjects. The abstractness of algorithms and data structures can pose significant learning barriers, making it difficult for students to visualize and apply concepts in a practical context.

Lectures, often employing the chalk-and-talk approach, provide a direct but sometimes static method of instruction. Educators present algorithmic concepts and operations of data structures in a manner that, although informative, may not effectively demonstrate the dynamic nature of these topics. Static code examples in textbooks offer much information but lack interactivity, limiting students' ability to experiment with algorithms and observe their behavior in various scenarios (Fincher & Petre, 2004).

Laboratory sessions are commonly used to complement lectures and textbook learning and bridge the gap between theory and practice. These sessions enable students to implement algorithms and manipulate data structures firsthand. However, they may not always succeed in highlighting the efficiency and real-time behavior of algorithms under different conditions, leaving some educational needs unmet (Simon et al., 2006).

With the advent of digital education technologies, there has been a gradual shift toward incorporating more dynamic and interactive tools into the curriculum. Visualization software and interactive platforms have emerged, allowing for the simulation of algorithm processes and data structure manipulations in real-time (Naps et al., 2002).

Integrating Algorithm Visualizations (AVs) has significantly enhanced student learning outcomes. Figure 1 investigates the impact of AVs on learning the **MergeSort** algorithm, which demonstrated a substantial improvement in students' comprehension when AVs were utilized. Prior to the intervention, both the control group (using traditional text-based materials) and the experimental group (using AVs) showed similar levels of understanding. However, post-intervention assessments revealed that the group exposed to AVs achieved a markedly higher average score (74%) compared to the text-based group (43%) (Hansen et al., 2002).

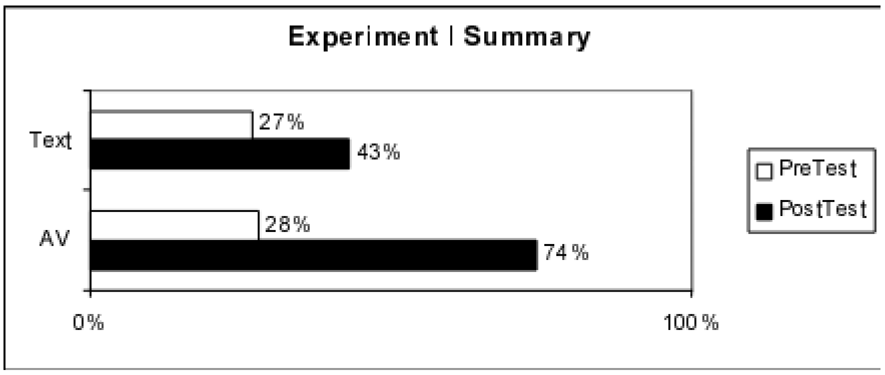


Fig. 1. Shows the overall experiment summary (Hansen et al., 2002, p. 14).

Nevertheless, traditional teaching methods continue to form the backbone of computer science education, providing essential knowledge and a basis for further exploration. Integrating these time-tested methods with innovative digital tools reflects an evolving educational paradigm, aiming to accommodate diverse learning styles and enhance the comprehension of complex computer science concepts (Guo, 2013).

2.3 Emergence of Algorithm Visualizers

The emergence of algorithm visualizers marks a significant evolution in the pedagogical tools available for computer science education. These digital platforms transform abstract data structures and algorithmic processes into interactive visual simulations, offering an innovative approach to learning and understanding complex concepts. This shift towards visual learning aids addresses the limitations of traditional teaching methods, providing a dynamic and engaging educational experience that caters to diverse learning styles.

Algorithm visualizers enable students to observe the step-by-step execution of algorithms, allowing for a deeper comprehension of their operational mechanics and efficiency in real time. This visualization clarifies complex algorithms and fosters an intuitive understanding by linking theoretical concepts to concrete visual outcomes. Grissom et al. (2003) highlighted the potential of visualization tools to improve learning outcomes by enabling students to see the immediate impact of code execution, thus bridging the gap between abstract theory and practical application.

Visualization software like **Algorithms and Data Structures**, **Algo**, and the Python Tutor offers interactive features enabling learners to manipulate data inputs and observe algorithm behavior, promoting experiential learning in line with constructivist theory (Piaget, 1970). These tools cover various algorithms, including sorting, searching, and graph algorithms, enhancing problem-solving skills and algorithmic thinking. Moreover, algorithm visualizers support collaborative and flipped classroom models by facilitating group discussions and interactive sessions, fostering peer-to-peer interaction and knowledge sharing, thus enriching the learning experience (Lahti et al., 2014).

2.4 Evaluation of Algorithm Visualizers

Evaluating algorithm visualizers as educational tools has garnered significant attention within the computer science education community. These visualizers aim to enhance learning outcomes and deepen students' understanding of complex algorithmic concepts by offering interactive and graphical representations of algorithms in action. However, the effectiveness of these tools is contingent upon various factors, including their design, usability, and the pedagogical strategies employed in their integration into the curriculum.

Research studies have evaluated the impact of algorithm visualizers on student learning with mixed results. Hundhausen et al. (2002) conducted empirical studies to assess the educational benefits of algorithm visualization (AV) tools. Their findings suggest that while AV tools can improve student performance in certain contexts, their effectiveness is highly dependent on how they are used in teaching and learning. This indicates that merely having access to visualizers is not enough; effective instructional support and integration into the learning activities are crucial for maximizing their benefits.

One key factor in successfully using algorithm visualizers is the level of engagement they foster among students. Naps et al. (2002) argue that for visualizers to be effective, they must support exploratory learning, allowing students to manipulate data and control the visualization process. This active engagement with the tool encourages deeper cognitive processing, which is essential for learning complex concepts.

Another aspect of evaluation focuses on the cognitive load imposed by the visualizers. Excessive information or overly complex visualizations can overwhelm students, detracting from the learning experience. Stasko et al. (1993) highlight the importance of designing visualizers that present information clearly and concisely, minimizing unnecessary cognitive load while still providing insightful visual representations of algorithmic processes.

The pedagogical context in which visualizers are used also plays a critical role in their effectiveness. Urquiza-Fuentes and Velázquez-Iturbide (2009) emphasize the need for integrating visualizers into a well-structured pedagogical framework that includes clear learning objectives, guided exploration, and reflective activities. This structured approach ensures that students are not just passive observers of the visualization but active learners, engaging with the material meaningfully.

Figure 2 presents a comparative analysis of AlgoLearn with two other existing projects across various evaluation metrics. AlgoLearn shares considerable similarities with 'Algorithms and Data Structures,' as evidenced by its commendable rating of 4.5 stars on Google Play and over 100,000 downloads. However, despite its superiority in several aspects compared to 'Algo,' our application does present certain limitations. Notably, AlgoLearn lacks comprehensive accessibility features such as support for dark themes, color blindness options, and interactive sound cues, which restricts usability for visually impaired users. Additionally, the application does not support interactive functionalities like zooming in and out. These omissions, primarily due to time constraints during development, slightly mar an otherwise robust application. Furthermore, compared to its competitors, AlgoLearn offers a narrower range of algorithms, reflecting a limitation that future updates could address to enhance its educational utility.

Applications / Criteria	AlgoLearn	Algorithms and Data Structures	Algo
User Interface (design)	4/5	3.5/5	3/5
Visualization features	4.5/5	4/5	2/5
Customization options	4.5/5	3/5	1/5
Engagement	4/5	4.5/5	2/5
Performance	4/5	4/5	5/5
Documentation	3/5	4/5	2/5
Variety of Algorithms	2/5	5/5	4/5
Accessibility Features	2/5	4/5	3.5/5

Fig. 2. Shows an evaluation of AlgoLearn against other existing applications.

Chapter 3: Methodology and Technology of Choices

3.1 Methodology

3.1.1 Agile Development Practices

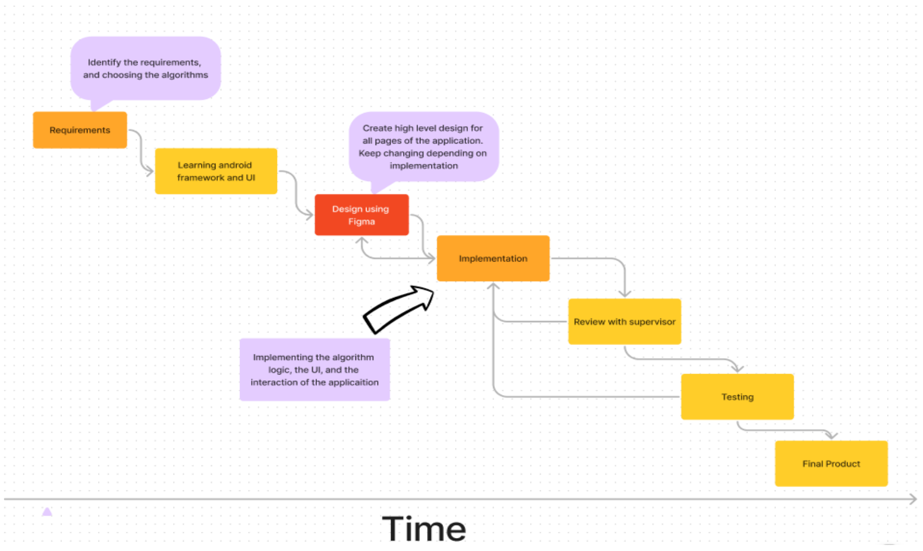
Agile development practices represent a group of methodologies in software development that emphasize incremental progress, collaboration, flexibility, and rapid adaptation to change. Originating from the Agile Manifesto, these practices promote a disciplined project management process that encourages frequent inspection and adaptation, self-organization, and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals (Beck et al., 2001). Agile methodologies such as Waterfall, Kanban, and Extreme Programming (XP) facilitate continuous iteration of development and testing throughout the lifecycle of a project.

3.1.2 Waterfall

Figure 3 explains the waterfall model employed for this project due to its orderly sequence, which is especially suitable for fixed and well-understood requirements. The project began with defining the software's goals and algorithms in the "Requirements" phase. Knowledge of the Android framework and UI principles was then acquired.

In the next Phase, initial high-level interface designs were created using Figma, with room for adjustments as development progressed. Coding and implementation followed, bringing the design to life and integrating the application's core logic.

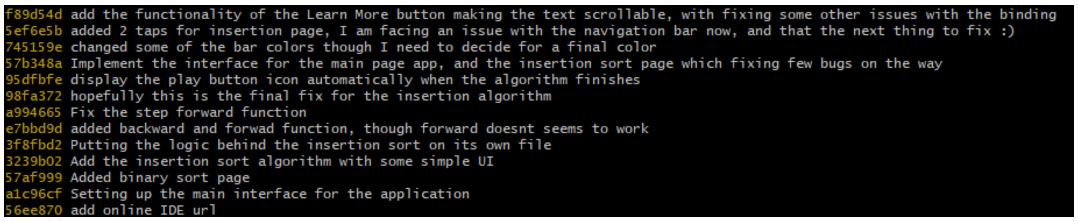
Periodic reviews with a supervisor ensured alignment with project goals, and testing was conducted to guarantee functionality and reliability. Despite the linear path of the Waterfall model, a feedback loop allowed for design iteration influenced by implementation discoveries.


Identify the requirements, and choosing the algorithms
Requirements
Learning android framework and UI
Create high level design for all pages of the application. Keep changing depending on implementation
Design using Figma
Implementation
Implementing the algorithm logic, the UI, and the interaction of the applicaition
Review with supervisor
Testing
Final Product
Time
Fig. 3. Waterfall diagram for project management

3.1.3 Git

Git is a vital distributed version control system for modern software development, ensuring speedy, reliable, and collaborative code management. It enables multiple developers to work concurrently on the same project without conflicts and maintains a detailed history of code changes. Git's robust features, including issue tracking and precise version control, safeguard project integrity and support efficient development workflows.

As illustrated in Figure 4, Git served as the primary tool for monitoring the project's progression. The code repository was hosted remotely on GitHub. Additionally, Git branches were employed to evaluate new functionalities, ensuring that they did not interfere with the main branch.



```
f89d54d add the functionality of the Learn More button making the text scrollable, with fixing some other issues with the binding
5ef6e5b added 2 taps for insertion page, I am facing an issue with the navigation bar now, and that the next thing to fix :)
745159e changed some of the bar colors though I need to decide for a final color
57b348a Implement the interface for the main page app, and the insertion sort page which fixing few bugs on the way
95dfbfe display the play button icon automatically when the algorithm finishes
98fa372 hopefully this is the final fix for the insertion algorithm
a994665 Fix the step forward function
e7bbd9d added backward and forwad function, though forward doesnt seems to work
3f8fbd2 Putting the logic behind the insertion sort on its own file
3239b02 Add the insertion sort algorithm with some simple UI
57af999 Added binary sort page
a1c96cf Setting up the main interface for the application
56ee870 add online IDE url
```

Fig. 4. Git commit history.

3.2 Choices of Technology

3.2.1 Java Programming Language

Java, a class-based, object-oriented programming language, is designed to have as few implementation dependencies as possible. Since its release by Sun Microsystems in 1995, Java has become one of the most popular programming languages, especially in enterprise environments and Android app development. Its widespread adoption is attributed to its robustness, security features, cross-platform capabilities through the Java Virtual Machine (JVM), and ease of use for building large-scale applications (Gosling et al., 2000).

In recent years, Kotlin, developed by JetBrains, has emerged as an alternative to Java for Android development. Google has officially supported Kotlin since 2017. Kotlin offers several modern features, such as null safety, extension functions, and coroutines for asynchronous programming, which can lead to more concise and expressive code compared to Java. Despite Kotlin's growing popularity and its advantages for new projects, Java continues to be extensively used in existing applications and by developers with a significant investment in the Java ecosystem.

Java was selected for this project due to its established presence as the leading programming language in Android development environments. Specifically, the industry's continued preference for Java, especially in legacy applications and among developers with extensive experience in Java-based development, played a crucial role in this decision. Java's comprehensive ecosystem, abundant libraries, and robust community support system provide essential tools for addressing the complexities of app development. Although Kotlin offers modern programming features that could enhance development efficiency, the established familiarity with Java, coupled with its demonstrated stability and performance in Android development, warranted its selection for the project.

3.2.2 Android Framework and Android Studio

The Android Framework offers a robust foundation for building mobile applications specifically for the Android operating system. It provides a rich set of pre-built UI components, a comprehensive development environment, and the necessary APIs to create interactive and immersive applications. With its open-source nature, the Android Framework enables developers to access a wide array of tools and libraries to streamline the development process from conception to deployment.

The project selected the Android Framework and Android Studio due to their robust features and efficient development environment. The Android Framework offers comprehensive support for mobile app development, while Android Studio provides an ideal platform with its extensive documentation, tutorials, and community support.

The Android operating system is structured around the fundamental component known as an "Activity." An Activity represents a single screen with a user interface. In essence, each Activity forms a page within the application. This modular approach allows developers to build applications with multiple screens, each dedicated to a specific purpose, ensuring a cohesive user experience. The UI structure within these Activities is versatile and hierarchical, allowing for complex layouts to be constructed from reusable components, which can be tailored to fit various device configurations and orientations; refer to figure X in the appendix that represents the activity lifecycle.

Android applications follow a structured folder organization to manage project resources effectively. Key folders include **src/** for Java source files**, res/** for non-code resources like XML layouts and bitmap images, **AndroidManifest.xm**l for declaring application components and permissions, **assets/** for raw file types, and **gradle/** for Gradle build system configurations.

3.2.3 Gradle and JUNG Library

The Android Build System is central to the development process for Android applications, facilitating the conversion of source code into Android Package (APK) files that can be executed on Android devices. Gradle, the official build automation tool adopted by Android, plays a pivotal role in this system. It is known for its flexibility, scalability, and dependency management capabilities. Gradle allows developers to define custom build configurations that include varying dependencies, build variants, and specific signing configurations, which makes it easier to manage complex projects and ensures a streamlined development and deployment process. Learning Gradle is essential for Android developers to build and deploy apps efficiently and understand how Android applications are structured and compiled.

The project integrates essential libraries through Gradle dependencies to improve the Android application's functionality and user interface design. These libraries offer tools and components vital for modern Android development, including support for material design UI components, constraint layout for flexible UI design, and testing frameworks for ensuring application reliability; refer to Figure 22 in the appendix, which represents all included libraries.

On the other hand, the Java Universal Network/Graph Framework (JUNG) is a software library that provides a familiar and extendable language for modeling, analyzing, and visualizing data that can be represented as a graph or network. In this project, JUNG plays a crucial role in implementing graph-based algorithms such as Dijkstra's algorithm for shortest path finding and DFS; the library allows users with an internal custom modification to manipulate graphs by adding weights, edges, and moving vertices. The choice of JUNG is motivated by its comprehensive collection of algorithms for network analysis, ease of integration with Java applications, and its capabilities for graph visualization. These features of JUNG have been instrumental in facilitating the visualization of algorithmic processes, making the application more interactive, and enhancing the learning experience for users.

3.2.4 Figma

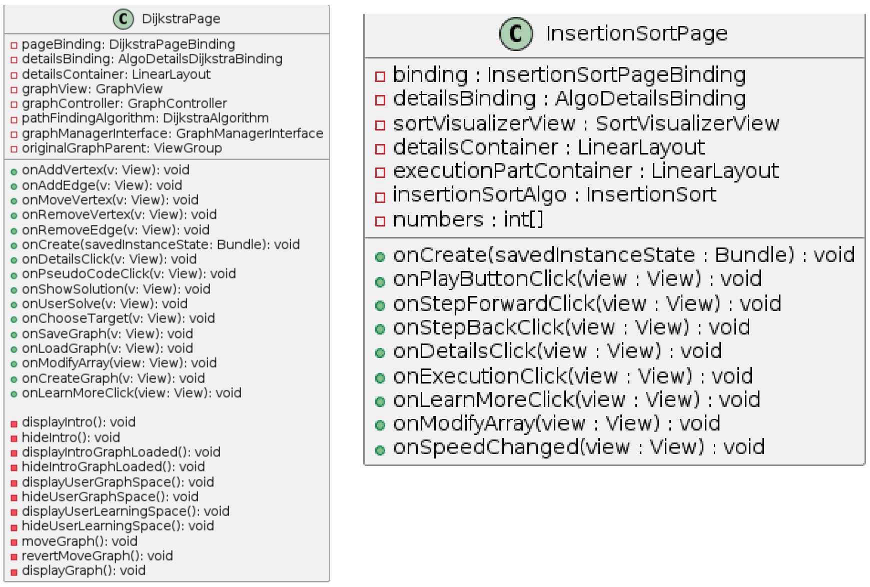
Figma is a powerful, cloud-based design tool that has significantly transformed the user interface (UI) and user experience (UX) design landscape. Its comprehensive toolkit allows for the creation of detailed wireframes, prototypes, and high-fidelity designs, making it an important resource in modern digital product development.

In AlgoLearn, Figma was utilized to design the UI, allowing for visualization and iteration of layout and functionality before starting to code. Collaborative design with supervisors facilitated the process, aligning with agile practices by enabling rapid feedback. Integration with **DHWise - Figma to Code** further streamlined development by importing Figma designs into Android Studio as functional XML code, facilitating the translation of design concepts into UI components.

3.3 Design

3.1 Architectural Design and Code Structure

AlgoLearn is architected using object-oriented principles, ensuring a modular, maintainable, and scalable codebase. Figure 5 shows an example of the architectural style that has been used in the application. The project’s structure reflects a clear and logical organization, adhering to Android development best practices and facilitating future enhancements or modifications.



```
DijkstraPage
pageBinding: DijkstraPageBinding
detailsBinding: AlgoDetailsDijkstraBinding
detailsContainer: LinearLayout
graphView: GraphView
graphController: GraphController
pathFindingAlgorithm: DijkstraAlgorithm
graphManagerInterface: GraphManagerInterface
originalGraphParent: ViewGroup
onAddVertex(v: View): void
onAddEdge(v: View): void
onMoveVertex(v: View): void
onRemoveVertex(v: View): void
onRemoveEdge(v: View): void
onCreate(savedInstanceState: Bundle): void
onDetailsClick(v: View): void
onPseudoCodeClick(v: View): void
onShowSolution(v: View): void
onUserSolve(v: View): void
onChooseTarget(v: View): void
onSaveGraph(v: View): void
onLoadGraph(v: View): void
onModifyArray(view: View): void
onCreateGraph(v: View): void
onLearnMoreClick(view: View): void

displayIntro(): void
hideIntro(): void
displayIntroGraphLoaded(): void
hideIntroGraphLoaded(): void
displayUserGraphSpace(): void
hideUserGraphSpace(): void
displayUserLearningSpace(): void
hideUserLearningSpace(): void
moveGraph(): void
revertMoveGraph(): void
displayGraph(): void
```

```
InsertionSortPage
binding : InsertionSortPageBinding
detailsBinding : AlgoDetailsBinding
sortVisualizerView : SortVisualizerView
detailsContainer : LinearLayout
executionPartContainer : LinearLayout
insertionSortAlgo : InsertionSort
numbers : int[]
onCreate(savedInstanceState : Bundle) : void
onPlayButtonClick(view : View) : void
onStepForwardClick(view : View) : void
onStepBackClick(view : View) : void
onDetailsClick(view : View) : void
onExecutionClick(view : View) : void
onLearnMoreClick(view : View) : void
onModifyArray(view : View) : void
onSpeedChanged(view : View) : void
```

Fig. 5. illustrates class diagrams for the selected Android activities.

Activity and XML File Pairing: Each screen within the application corresponds to a distinct Activity, with naming conventions that highlight the algorithmic focus; for example, "DijkstraPage" alongside its UI representation in an XML file, "dijkstra_page.xml." Figure 6 presents an overview of the interaction between logical components and user interface elements. The "DijkstraPage" acts as the logical controller, managing button presses and additional functionalities, and is connected to its visual representation outlined in "dijkstra_page.xml.” This central XML file branches into two separate XML elements, "algo_details.xml" for algorithm descriptions and "pseudo_code.xml" for displaying the pseudocode.

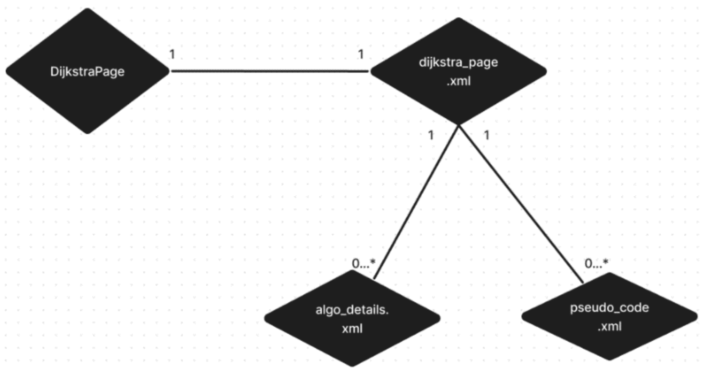


Fig. 6. Shows the relation between the logic and the UI.

Architectural Style: The application follows a well-defined architectural style that promotes the separation of concerns and optimizes code readability. Specifically, it employs the Model-View-Controller (MVC) pattern, which segments an application into three core components. The Model component manages the algorithms and their data structures, ensuring independence from the user interface. The View presents the model's data to the user, comprising the graphical elements and visualizations that interactively represent algorithm processes. The Controller interprets user inputs, translating them into actions to be performed by the Model or updates to be reflected in the View. Furthermore, the application incorporates the Observer pattern, enabling components to subscribe to and be notified of events. This pattern is instrumental in updating the UI in response to changes in the algorithm's state or data, fostering a dynamic and responsive learning environment. "**PathUpdateListener**" is an interface that serves as an example in AlgoLearn, designed to allow real-time updates across the UI in response to algorithmic computations.

Figure 7 provides a detailed flowchart illustrating the architectural design and explains the dynamic interactions among various components within the "DijkstraPage." This visualization helps us understand how each element contributes to the overall functionality and structure of the page.

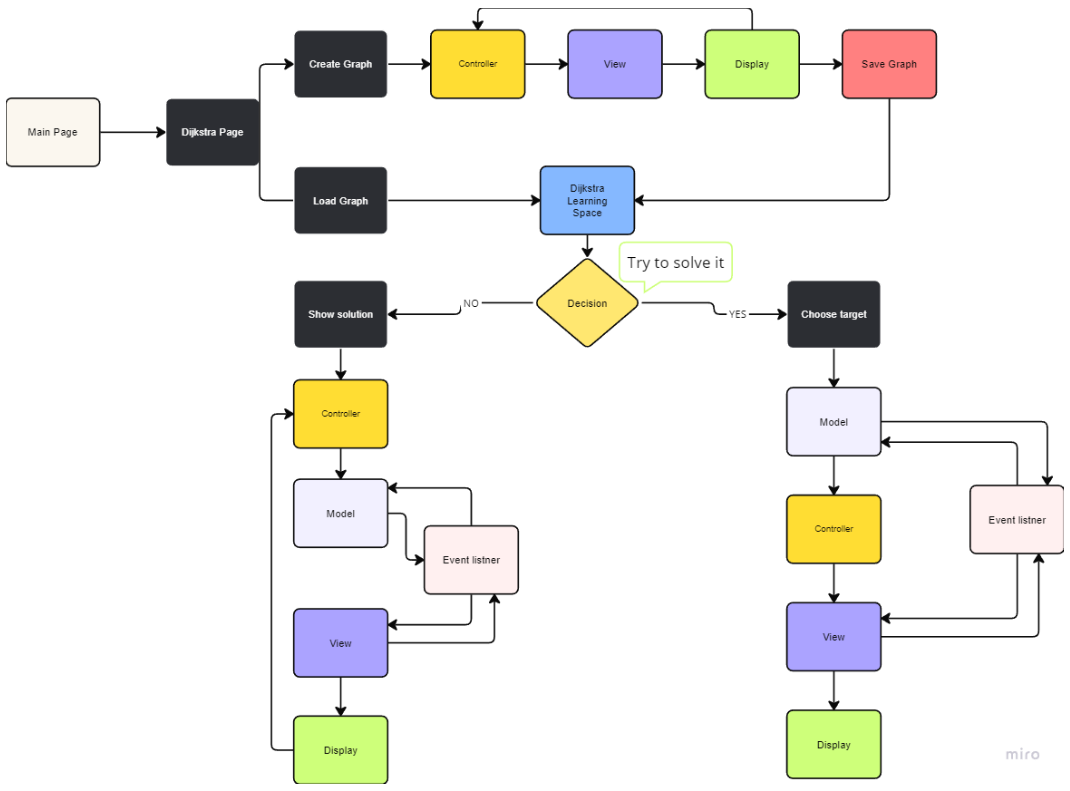
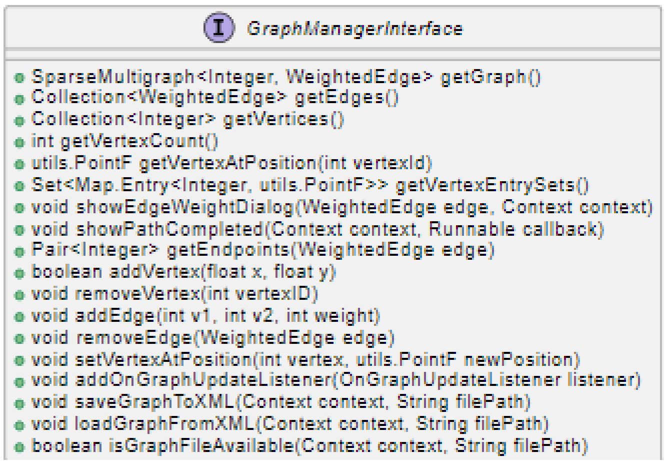


Fig. 7. Flowchart that describes the architecture style for the Dijkstra page

Graph Manipulation Interface: The introduction of a "GraphManagerInterface" was strategically implemented to accommodate future demands for flexibility. This interface abstracts graph operations, rendering the system independent of the specific underlying data structure, whether it involves in-memory objects or a database, as depicted in Figure 8. This prudent decision facilitates seamless adaptation and scalability, ensuring that AlgoLearn can evolve with minimal requirement for significant rewrites.



```
I GraphManagerInterface

SparseMultigraph<Integer, WeightedEdge> getGraph()
Collection<WeightedEdge> getEdges()
Collection<Integer> getVertices()
int getVertexCount()
utils.PointF getVertexAtPosition(int vertexId)
Set<Map.Entry<Integer, utils.PointF>> getVertexEntrySets()
void showEdgeWeightDialog(WeightedEdge edge, Context context)
void showPathCompleted(Context context, Runnable callback)
Pair<Integer> getEndpoints(WeightedEdge edge)
boolean addVertex(float x, float y)
void removeVertex(int vertexID)
void addEdge(int v1, int v2, int weight)
void removeEdge(WeightedEdge edge)
void setVertexAtPosition(int vertex, utils.PointF newPosition)
void addOnGraphUpdateListener(OnGraphUpdateListener listener)
void saveGraphToXML(Context context, String filePath)
void loadGraphFromXML(Context context, String filePath)
boolean isGraphFileAvailable(Context context, String filePath)
```

Fig. 8. Class diagram.

Organized Code and Resources: Algorithmic logic is in a dedicated "Algorithms" folder. Utility functions are centrally located in a utils package, serving various application parts and promoting code reuse.

3.2 General and Interactive UI Features

Since the project's beginning, the application's graphical user interface (GUI) has undergone multiple developments. This development shows a dedication to enhancing usability, aesthetics, and the user learning experience. From the initial wireframes to the final design, the GUI has been refined to be functional and engaging. It has been made more intuitive, making it easier for users to interact with the application.

The design centers around the AlgoLearn logo, a consistent brand identifier at the screen's top. Below the logo, strategically positioned buttons represent each covered algorithm, accompanied by descriptive images visually representing the algorithm's concept. Tversky et al. (2002) highlight the importance of visual cues in guiding user attention and aiding memory, suggesting that such elements can make learning environments more efficient by reducing cognitive load. The strategic placement and design of the buttons in AlgoLearn, paired with visual representations of the algorithms, are rooted in these findings, aiming to demystify complex concepts and enhance the user's learning experience.

Figure 9 illustrates the progression of the UI development process, tracing its evolution from the initial state to the final state over time. This sequential representation provides a detailed visualization of the interface enhancements and modifications implemented throughout the development cycle. It offers insight into the design decisions and improvements contributing to the final layout and functionality.

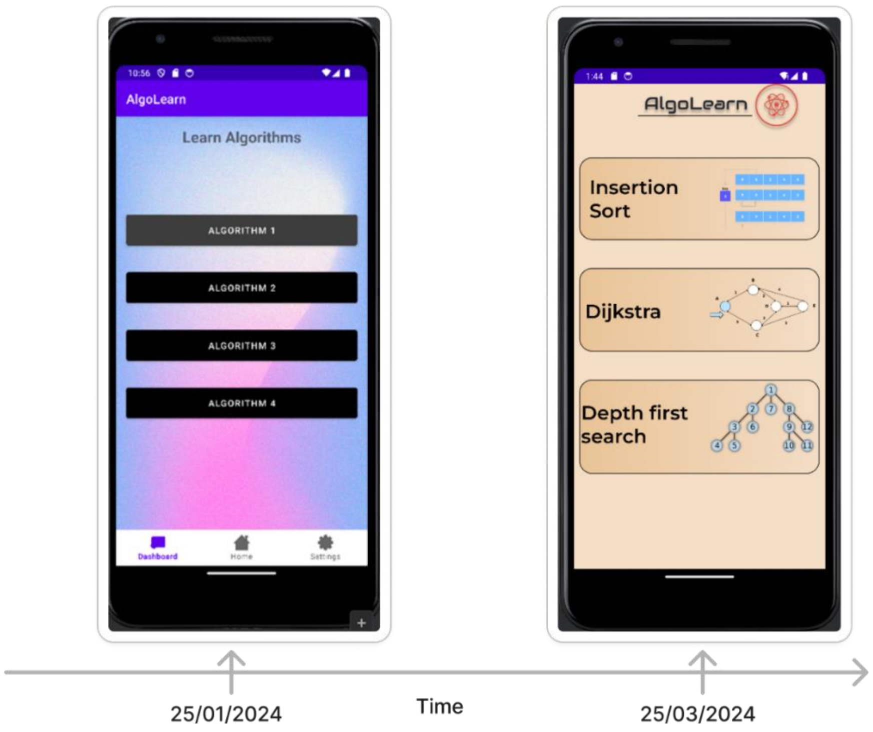


Fig. 9. Shows the main menu UI development process.

Chapter 4: Implementation Details

4.1 Dijkstra's Algorithm

4.1.1 Implementation Details

The algorithm's implementation showcases a classic approach to solving the shortest path problem in a weighted graph. This algorithm is essential for understanding fundamental concepts in graph theory and its application in routing and navigation problems.

Algorithm Flow:

Initialization: Each vertex is initialized with an infinite distance (except the source vertex, set to 0) to indicate that the shortest paths are not yet calculated.

Exploration: The algorithm iteratively explores vertices, starting from the source. For each vertex “u” polled from the priority queue, it examines all outgoing edges to adjacent vertices v.

Relaxation: If a shorter path to v through u is discovered (**distanceThroughU**), the algorithm updates v's distance and predecessor. This step is known as relaxation and is central to Dijkstra's algorithm.

Path Reconstruction: Once all vertices are explored, the algorithm can reconstruct the shortest path from the source to any target vertex by following the predecessors from the target back to the source.

Core Components:

Figure 10 summarizes the core components of the class responsible for the Dijkstra algorithm logic. Each component is described in detail below:

Graph Representation: The graph, represented by **Graph<Integer, WeightedEdge>**, consists of vertices (Integer) and edges (WeightedEdge), illustrating the interconnected network over which the shortest paths are calculated.

Distance and Predecessor Tracking: Two critical HashMaps, distances and predecessors, store the minimum distance from the source to each vertex and the previous vertex in the optimal path, respectively.

Unvisited Set: An unvisited HashSet keeps track of all vertices not finalized for the shortest path, ensuring the algorithm progresses through the graph efficiently.

Priority Queue: The algorithm employs a PriorityQueue<Integer> to prioritize vertices based on their current shortest distance from the source, enabling the selection of the next vertex to visit greedily.

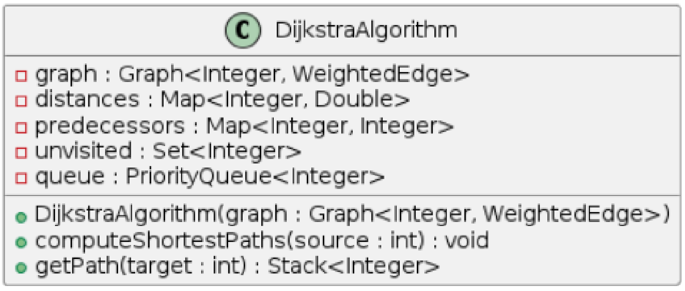
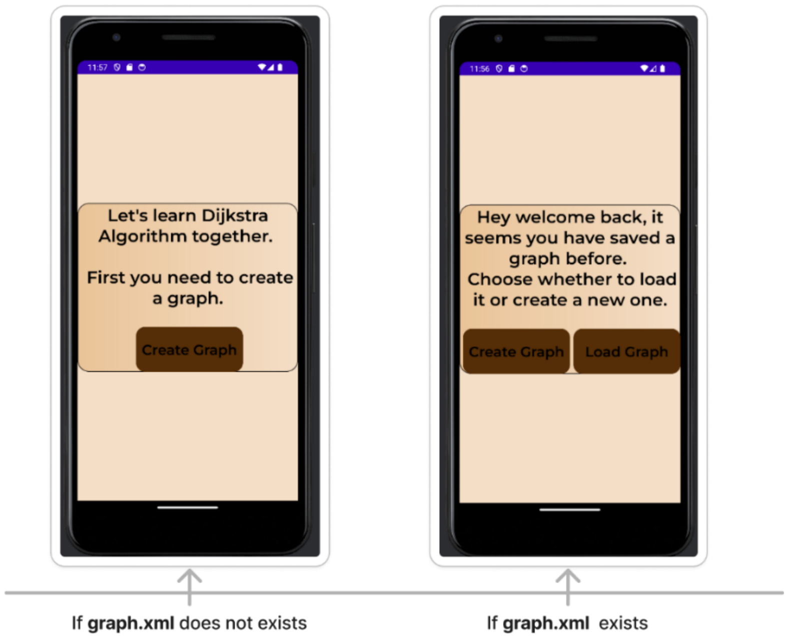

C DijkstraAlgorithm
graph : Graph<Integer, WeightedEdge>
distances : Map<Integer, Double>
predecessors : Map<Integer, Integer>
unvisited : Set<Integer>
queue : PriorityQueue<Integer>
DijkstraAlgorithm(graph : Graph<Integer, WeightedEdge>)
computeShortestPaths(source : int) : void
getPath(target : int) : Stack<Integer>

Fig. 10. Class diagram for Dijkstra logic.

4.1.2 Visualization and UI Design

User interface design plays a crucial role in facilitating an immersive learning experience. Upon accessing Dijkstra's algorithm feature, users are immediately engaged through a dialog prompt asking whether they wish to load a pre-existing graph or create a new one. This functionality is encapsulated in the **onLoadGraph(View v**) method, which searches for an existing graph stored in the internal memory under **savedGraphs/graph.xml**. Figure 11 illustrates the different scenarios that occur upon launching the application, depending on whether a graph is already stored in the device memory.


Let's learn Dijkstra Algorithm together.

First you need to create a graph.

Hey welcome back, it seems you have saved a graph before.
Choose whether to load it or create a new one.

Fig. 11. Shows the two different scenarios upon launching the Dijkstra Algorithm.

If the user opts to construct a new graph, they are transitioned to an interactive canvas designed for graph construction, as shown in Figure 12. This environment has buttons labeled to perform actions such as adding, moving, and removing vertices, connecting vertices with edges, and assigning weights to these edges. Following the graph's customization, the **onSaveGraph(View v)** method is pivotal in validating and persisting the graph's structure as an XML file within the **savedGraphs** directory of Android's internal memory. This method ensures the preservation of user-created graphs and enables later retrieval and exploration.

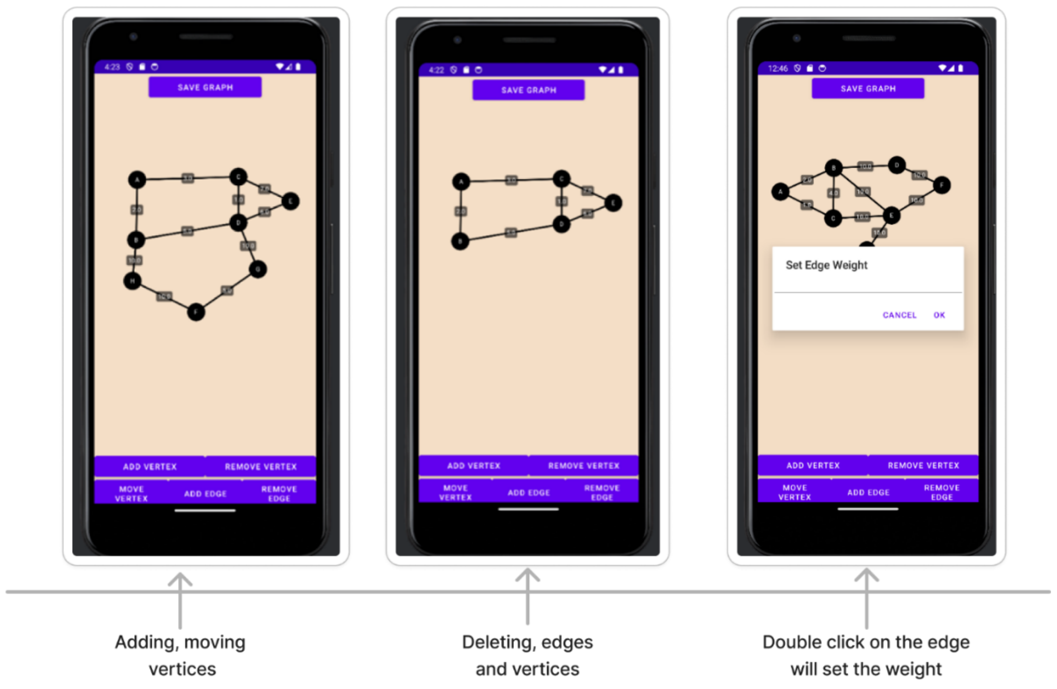
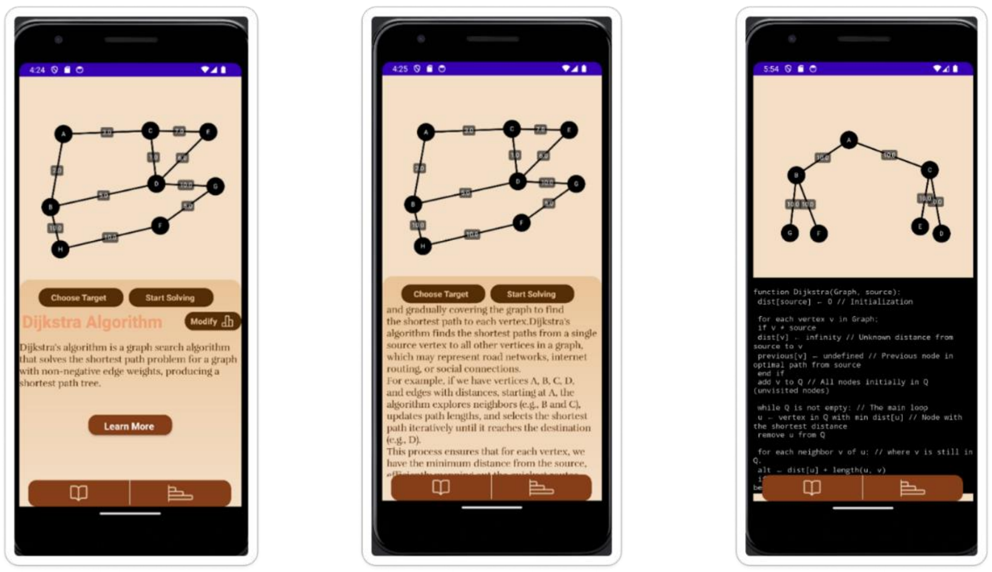


Fig. 12. Displays the canvas for creating a graph.

The final phase of the user journey showcases the custom graph on the upper half of the screen with a comprehensive explanation of Dijkstra's algorithm on the lower half as shown in Figure 13. This layout is deliberately chosen to facilitate a direct correlation between theoretical concepts and their practical application, enhancing the educational value of the algorithm's visualization.



Dijkstra's algorithm is a graph search algorithm that solves the shortest path problem for a graph with non-negative edge weights, producing a shortest path tree.

and gradually covering the graph to find the shortest path to each vertex.Dijkstra's algorithm finds the shortest paths from a single source vertex to all other vertices in a graph, which may represent road networks, internet routing, or social connections.
For example, if we have vertices A, B, C, D, and edges with distances, starting at A, the algorithm explores neighbors (e.g., B and C), updates path lengths, and selects the shortest path iteratively until it reaches the destination (e.g., D).
This process ensures that for each vertex, we have the minimum distance from the source,

```
function Dijkstra(Graph, source):
 dist[source] ← 0 // Initialization

 for each vertex v in Graph:
 if v ≠ source
 dist[v] ← infinity // Unknown distance from
source to v
 previous[v] ← undefined // Previous node in
optimal path from source
 end if
 add v to Q // All nodes initially in Q
(unvisited nodes)

 while Q is not empty: // The main loop
 u ← vertex in Q with min dist[u] // Node with
the shortest distance
 remove u from Q

 for each neighbor v of u: // where v is still in
Q.
 alt ← dist[u] + length(u, v)
```

Fig. 13. Shows the user learning space of the algorithm.

4.1.3 User Interaction and Learning Experience

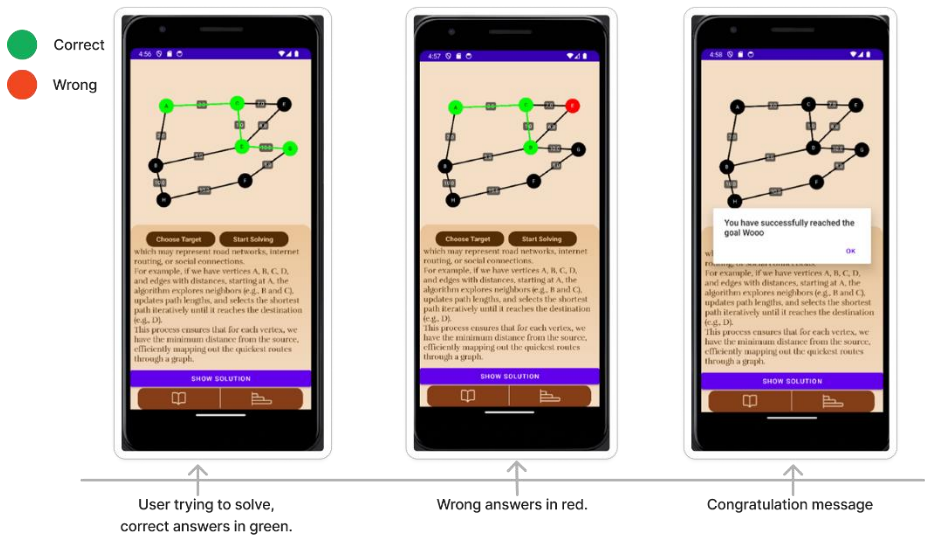
The journey begins with users having the option to either create a new graph or load an existing one, thereby establishing a flexible starting point for exploration. Figure 14 depicts the user learning environment, illustrating this initial interaction phase. At any stage of the process, users retain the capability to modify their graph by utilizing the **Modify** button, a feature that allows them to reassess and adjust their graph as needed.

A target node is chosen for Dijkstra's algorithm to compute the shortest path. This process, triggered by the "**Choose Target**" button, shifts users into an interactive problem-solving mode. Here, they leverage their knowledge of the algorithm to deduce the most efficient route to the chosen target. Initiating the challenge by pressing "**Start Solving**" transitions the application into a user-solving state. In this mode, participants actively engage by selecting the subsequent path they believe has the minimal weight.

Mistakes during this process are met with immediate feedback. Selecting an incorrect node results in a temporary red highlight, visually indicating the error and allowing users to reassess their choices. Conversely, correct selections are affirmed with a green highlight, both on the node and the connecting edges, visually reinforcing the correct understanding and application of the algorithm.

Achieving the goal of reaching the target node through the shortest path is celebrated with a congratulatory message, acknowledging the user's success and encouraging further exploration, which is pivotal in enhancing the learning experience.

The application offers a **Learn More** option for users facing challenges with complex graphs or seeking to solidify their understanding. This feature directs users to additional resources and, most notably, includes a **Show Solution** button. Upon selecting a new target node and initiating this function, the application automatically computes and displays the shortest path, providing a visual and interactive demonstration of Dijkstra's algorithm in action.



which may represent road networks, internet routing, or social connections.
For example, if we have vertices A, B, C, D, and edges with distances, starting at A, the algorithm explores neighbors (e.g., B and C), updates path lengths, and selects the shortest path iteratively until it reaches the destination (e.g., D).
This process ensures that for each vertex, we have the minimum distance from the source, efficiently mapping out the quickest routes through a graph.

You have successfully reached the goal Wooo

Fig. 14. Shows the user interaction.

4.2 Insertion Sort Algorithm

4.2.1 Implementation Details

The "InsertionSort" class is integrated within an Android activity and encapsulates the implementation of the insertion sort algorithm, utilizing a SortVisualizerView for visual representation. This algorithm is pivotal for understanding basic sorting mechanisms, which are fundamental to data organization and retrieval processes in computing.

Singleton Pattern: Ensures that only one instance of **InsertionSort** exists throughout the application lifecycle, providing a global point of access to it.

Algorithm Flow:

Initialization: Upon activation, the InsertionSort class initializes the sorting environment, including setting up the array of numbers to be sorted and preparing the visualizer and other UI elements.

Insertion: The sorting process begins at the second element, considering the first element as sorted. The current element is compared with those in the sorted section and inserted into the correct position. This step is repeated for each element until the array is fully sorted.

Visualization and Control: As sorting progresses, each operation (like swaps) is visually represented on the SortVisualizerView and tracked via a stack for possible backward navigation. Control flags such as isPaused, finished, and isForwardStepRunning manage the flow of the sorting process and UI interactions.

Core Components:

Figure 15 summarizes the core components of the class responsible for the insertion sort logic. Each component is described in detail below:

Singleton Pattern: This pattern ensures that only one instance of the InsertionSort class exists within the application, providing a global point of access and preventing multiple instantiations.

Thread Management: The sorting algorithm runs on a separate thread (sortingThread) to maintain UI responsiveness and prevent the UI from freezing during the sort operation.

UI Binding and Layout Management: This method utilizes a binding instance to interact with the XML layout elements and an execution container (execContainer) to visualize each step of the execution process.

Visualization Components: This feature features a custom view (visualizer) dedicated to the dynamic visual representation of the sorting process.

Data Structures for Operation Tracking: This method employs a stack (operations) to record each operation during the sorting process, enabling backward navigation through the sorting steps. An array (numbers) holds the integers that are to be sorted.

Control Flags: Implements control flags like isPaused, finished, and isForwardStepRunning to efficiently manage the sorting process flow and facilitate user interaction with the UI during the sorting process.

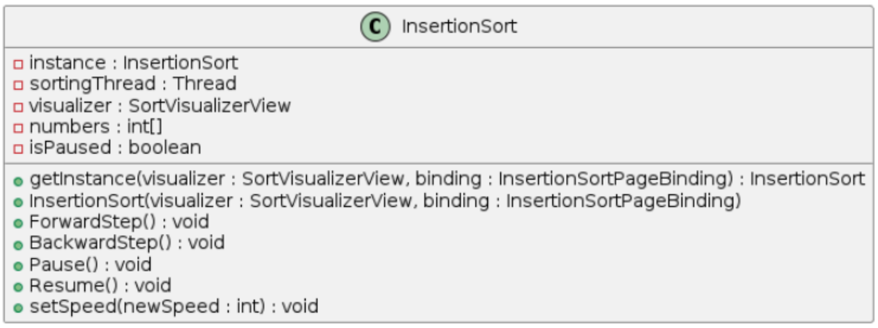

InsertionSort
instance : InsertionSort
sortingThread : Thread
visualizer : SortVisualizerView
numbers : int[]
isPaused : boolean
getInstance(visualizer : SortVisualizerView, binding : InsertionSortPageBinding) : InsertionSort
InsertionSort(visualizer : SortVisualizerView, binding : InsertionSortPageBinding)
ForwardStep() : void
BackwardStep() : void
Pause() : void
Resume() : void
setSpeed(newSpeed : int) : void

Fig. 15. Class diagram

4.2.2 Visualization and UI Design

Upon navigating to the insertion sort page, users are greeted with a visual representation of a default array, depicted as bars of varying heights corresponding to the elements' values which is clearly shown in Figure 16. This visual setup aims to make the abstract sorting process tangible and engaging.

The interface is equipped with several interactive controls to guide users through the sorting process:

Speed Control: Allows users to adjust the speed of the sorting animation, catering to their pace of learning.

Step Back/Forward: Enable users to navigate through the sorting steps backward or forward.

Pause/Start: A toggle button that lets users pause the sorting animation to analyze the current state or resume it to continue observing the sorting process.

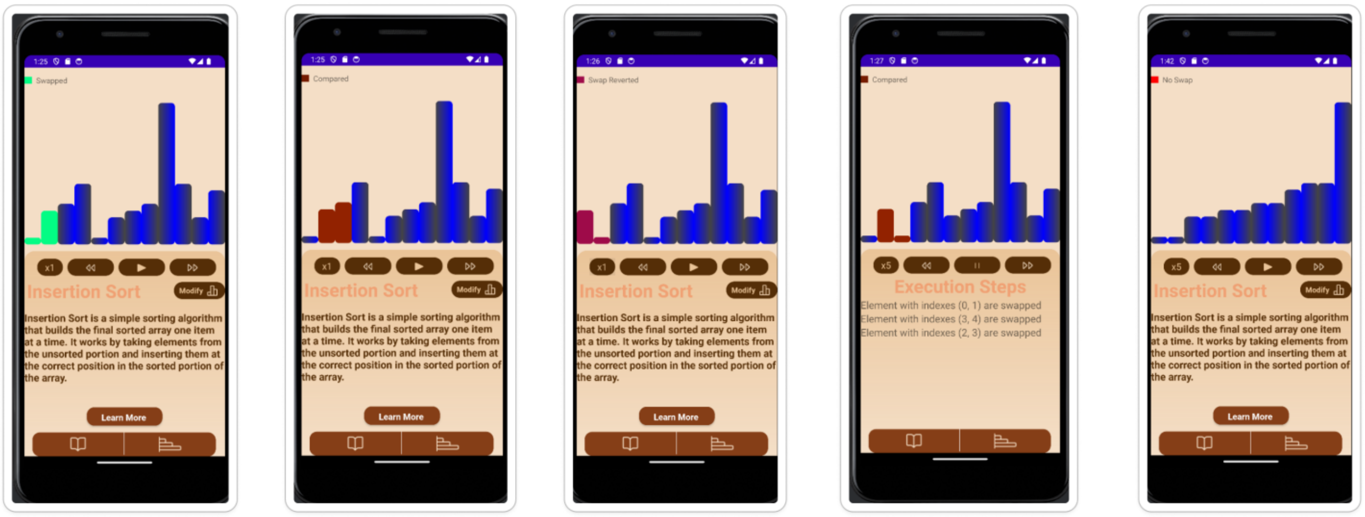
Modify: Presents users with the option to customize the array by entering a comma-separated list of numbers.

Learn More: Directs users to additional information about the insertion sort algorithm.

Execution Steps: Provides a visual log of each operation in the sorting process.

The visualizer employs color coding to indicate the status of elements during the sorting process:

Colors: grey marks the elements that are currently being compared, red indicates no swap, green highlights elements that have been swapped, and purple highlights elements that have been reverted.



Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It works by taking elements from the unsorted portion and inserting them at the correct position in the sorted portion of the array.

Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It works by taking elements from the unsorted portion and inserting them at the correct position in the sorted portion of the array.

Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It works by taking elements from the unsorted portion and inserting them at the correct position in the sorted portion of the array.

Element with indexes (0, 1) are swapped
Element with indexes (3, 4) are swapped
Element with indexes (2, 3) are swapped

Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It works by taking elements from the unsorted portion and inserting them at the correct position in the sorted portion of the array.

Fig. 16. Shows preview of the insertion sort page

4.2.3 User Interaction and Learning Experience

The Insertion Sort page offers users the ability to engage actively with the algorithm through customizable arrays, encouraging them to test various scenarios. The app guides users to predict and verify the next steps in the sorting process, enhancing their understanding and engagement. Tools to navigate the sorting steps allow for a detailed exploration of the algorithm, offering insights into each action. Visibility of execution steps connects theoretical knowledge with practical application, reinforcing learning. A feedback loop, through visual cues and the option to rewind steps, helps users identify and correct misunderstandings, fostering a deeper comprehension and application of the sorting algorithm. This interactive and reflective learning environment transforms users from passive viewers into active participants, deepening their understanding and confidence in algorithmic problem-solving.

4.3 Depth-first search

4.1.3.1 Implementation Details

The Depth-First Search (DFS) algorithm is a fundamental technique used in graph theory for traversing or searching through the nodes of a graph. Employed within the AlgoLearn project, this algorithm advances through each branch to its fullest extent before backtracking, facilitating an efficient exploration of graph structures. This approach not only underscores the algorithm's utility in illustrating graph theory but also enhances its application in educational settings.

Algorithm Flow:

Initialization: The traversal begins at a selected vertex, marking it as visited to prevent revisits and avoid potential infinite loops.

Exploration: The algorithm recursively visits each adjacent vertex that has yet to be visited, exploring as deeply as possible along each branch before backtracking.

Backtracking: Upon reaching a vertex with no further unvisited adjacent vertices, the algorithm backtracks to explore new paths from previously visited vertices.

Termination: The traversal ends when all vertices accessible from the starting point have been visited, ensuring a complete component exploration.

Core Components:

Figure 17 summarizes the core components of the class responsible for the Depth-First Search algorithm logic. Each component is described in detail below:

Graph Representation: Utilizes the same structure employed in the Dijkstra algorithm to represent the graph.

Visited Nodes Tracking: A Set named visited tracks the nodes that have been visited during the DFS traversal, which is crucial for preventing the algorithm from revisiting nodes and creating infinite loops.

Parent Tracking: A Map named parent links each vertex to its predecessor in the traversal path, facilitating the reconstruction of the traversal path for educational or analytical purposes.

Traversal Order: A Stack named traversalOrder captures the order in which nodes are visited, which is essential for understanding the algorithm's depth-first nature.

Visualization Integration: The DFS class is integrated with a GraphView object via the setGraphView method, allowing for real-time visualization of the DFS traversal on the UI. This visual representation highlights the exploratory steps and path of the algorithm, enhancing its educational value.

Traversal Algorithm: The dfsVisit method embodies the core logic of the DFS. It recursively explores adjacent nodes, marking them as visited, and logs the traversal order. Before visiting neighboring nodes, outgoing edges are sorted based on vertex positions to maintain a consistent exploration pattern, particularly beneficial for demonstrations.

Recording Every Step: The everyStep feature captures each vertex visit during the traversal, documenting both exploratory steps and backtracking actions, which are pivotal for a comprehensive understanding of the DFS process.

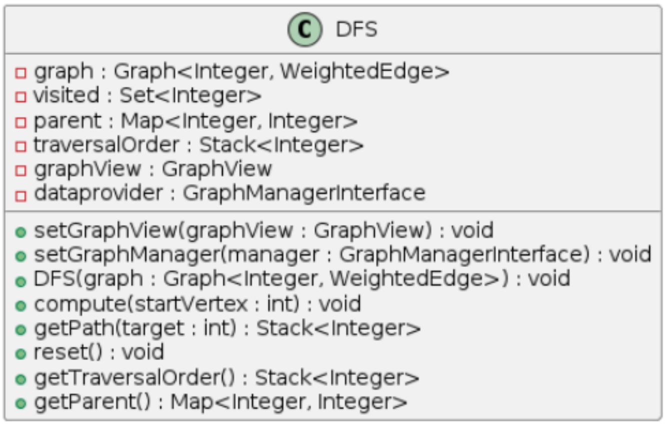

C DFS
graph : Graph<Integer, WeightedEdge>
visited : Set<Integer>
parent : Map<Integer, Integer>
traversalOrder : Stack<Integer>
graphView : GraphView
dataprovider : GraphManagerInterface
setGraphView(graphView : GraphView) : void
setGraphManager(manager : GraphManagerInterface) : void
DFS(graph : Graph<Integer, WeightedEdge>) : void
compute(startVertex : int) : void
getPath(target : int) : Stack<Integer>
reset() : void
getTraversalOrder() : Stack<Integer>
getParent() : Map<Integer, Integer>

Fig. 17. Class diagram.

4.3.2 Visualization and UI Design

The visualization design maintains a consistent user experience with the layout previously established for the Dijkstra algorithm. Upon selecting or creating a graph, users are presented with an interactive canvas on the upper half of the screen, while the lower half is dedicated to an insightful exposition of the DFS algorithm. This section can be expanded through the **Learn More** button, providing users with detailed explanations and illustrative examples to deepen their understanding. A navigation bar at the bottom of the screen offers access to a comprehensive description and the algorithm's pseudocode, facilitating a thorough comprehension of its inner workings.

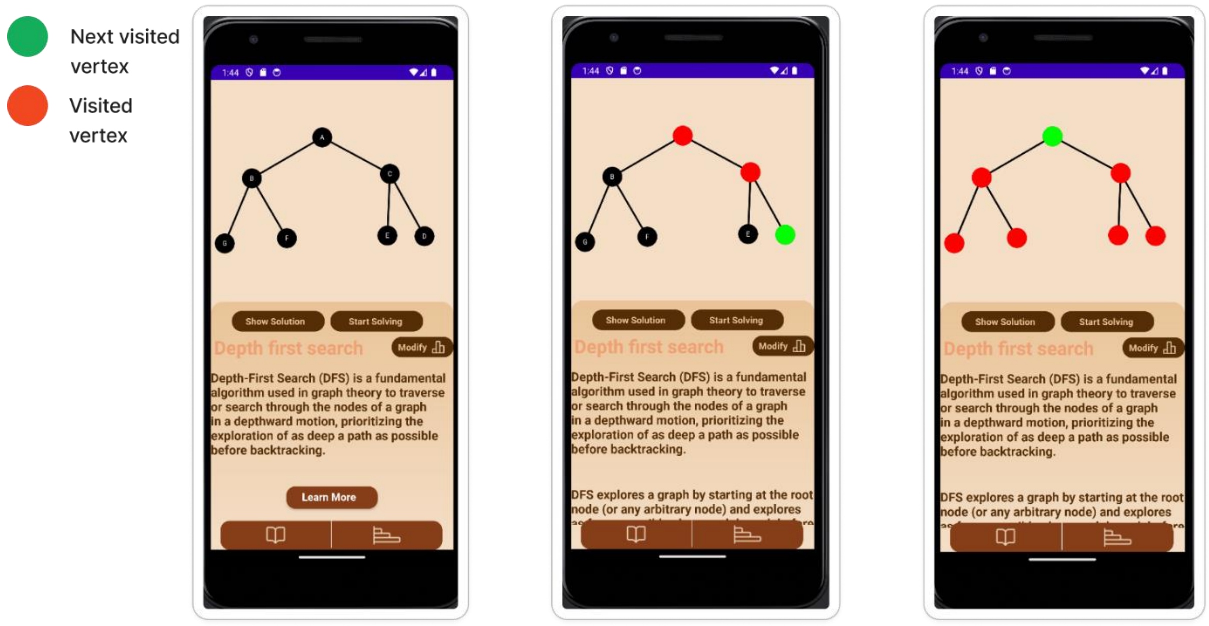
Similar to other algorithms featured in the application, the **Modify** button allows users to adjust the graph according to their learning needs, enabling a hands-on approach to exploring the algorithm's behavior under various graph configurations. Central to the user's engagement, the **Show Solution** button is strategically placed below the graph visualization, adjacent to the Start Solving button.

4.3.3 User Interaction and Learning Experience

The learning experience is structured to encourage active engagement and self-driven discovery, consistent with the educational approach applied across all algorithms featured. Users are invited to understand the algorithm through detailed descriptions and pseudocode, fostering a deeper comprehension of DFS's traversal mechanics as shown in Figure 18. The `**Show Solution**` demonstrates the algorithm's correct application to the graph, offering users an authoritative guide to DFS’s pathfinding strategy.

Simultaneously, the **Start Solving** button challenges users to apply their understanding by predicting the algorithm's traversal decisions. As users select vertices, correct choices are highlighted in red to indicate successful visits, while incorrect selections signal a misunderstanding of the algorithm's logic. This interactive learning model reinforces the user’s grasp of DFS and keeps engagement high through immediate feedback and the satisfaction of solving the traversal puzzle.

The educational journey culminates when all vertices have been correctly traversed, at this point, the application acknowledges the user's success with a congratulatory message.



Depth-First Search (DFS) is a fundamental algorithm used in graph theory to traverse or search through the nodes of a graph in a depthward motion, prioritizing the exploration of as deep a path as possible before backtracking.

Depth-First Search (DFS) is a fundamental algorithm used in graph theory to traverse or search through the nodes of a graph in a depthward motion, prioritizing the exploration of as deep a path as possible before backtracking.

DFS explores a graph by starting at the root node (or any arbitrary node) and explores

Depth-First Search (DFS) is a fundamental algorithm used in graph theory to traverse or search through the nodes of a graph in a depthward motion, prioritizing the exploration of as deep a path as possible before backtracking.

DFS explores a graph by starting at the root node (or any arbitrary node) and explores

Fig. 18. Shows a preview of the DFS algorithm.

Chapter 5: Evaluation

5.1 Unit Testing

Unit testing is an essential method of software testing where individual components of an application are tested in isolation from the rest of the system to ensure they operate correctly. In the AlgoLearn project, unit testing was employed as a fundamental testing technique, with JUnit serving as the primary testing library. The approach facilitated the examination of the application's algorithms—insertion sort, Dijkstra's, DFS, and UI ensuring their reliability and correctness.

Figure 19 illustrates a use case diagram, which served as a blueprint for the unit testing regimen of the Depth-First Search (DFS) algorithm. The testing structure ensured that each component was tested individually. The unit tests commenced with validating the 'Create Graph' and 'Load Graph' functionalities, verifying users' structural and logical integrity of graphs instantiated or ingressed. The algorithm's central logic was evaluated through a suite of tests, employing a different set of graph configurations to assess the robustness of the DFS process. Critical to this was the assessment of the 'Choose Next Vertex' function, which was benchmarked against pre-determined, correct traversals to ensure accuracy in real-time decision-making within the algorithm. This iterative verification persisted until all vertices were accounted for, which aligned with the DFS's completion condition. The DFS testing methodology extended beyond itself, serving as a standard protocol for evaluating all constituent components within the application.

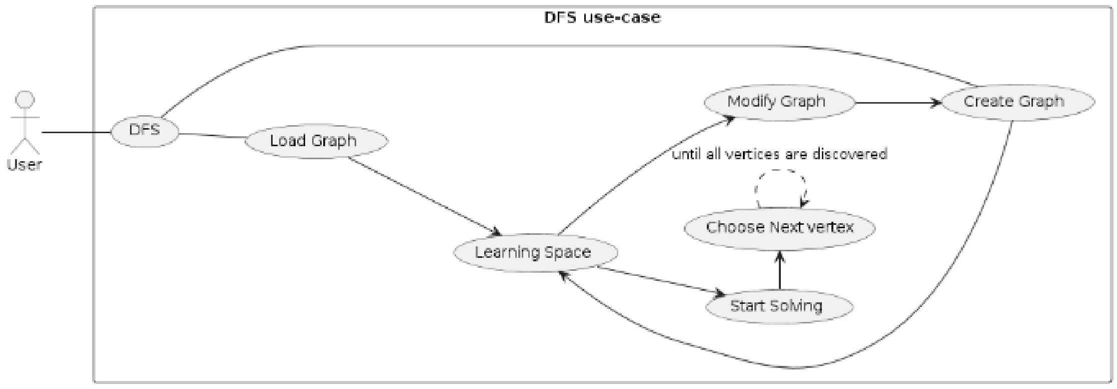


Fig. 19. Shows a use-case diagram for DFS.

The project incorporates 52 unit tests, each designed to scrutinize various functionalities across the application, focusing on the most granular components of the functions. Figure 20 presents an example of several selected tests. Refer to Heading H, 'Testing' in the Appendix, for a detailed view of these tests.

Test Name	Category	Test Case Description	Status
testInitialState	UI Visibility	Checks if initial visibility of elements matches expected startup state.	Pass
testOnPlayButtonClick	UI Interaction	Ensures play/pause button toggles visualizer state correctly.	Pass
testOnDetailsClick	UI Navigation	Verifies visibility changes and button interactions in details view.	Pass
testOnExecutionClick	Input Validation	Ensures only correct numerical input is accepted for array modification.	Pass

Fig. 20. Table shows a sample of the conducted unit tests.

5.2 Integration Testing

Integration testing is a software testing phase where individual units or components are combined and tested as a group to identify faults in their interactions. This testing ensures that the components function together as intended and that data flows accurately between them.

Integration testing for Android applications leverages Espresso and UI Automator to ensure components work seamlessly together. Espresso facilitates testing of in-app interactions by simulating user actions and verifying UI states, which is ideal for following flows outlined in sequence diagrams like the one provided for an Insertion Sort activity. UI Automator allows for broader tests involving multiple apps or system interactions. The sequence diagram provided in Figure 21 is a foundation for writing these tests, illustrating how users interact with the app and its logical processes. From there, tests are made.

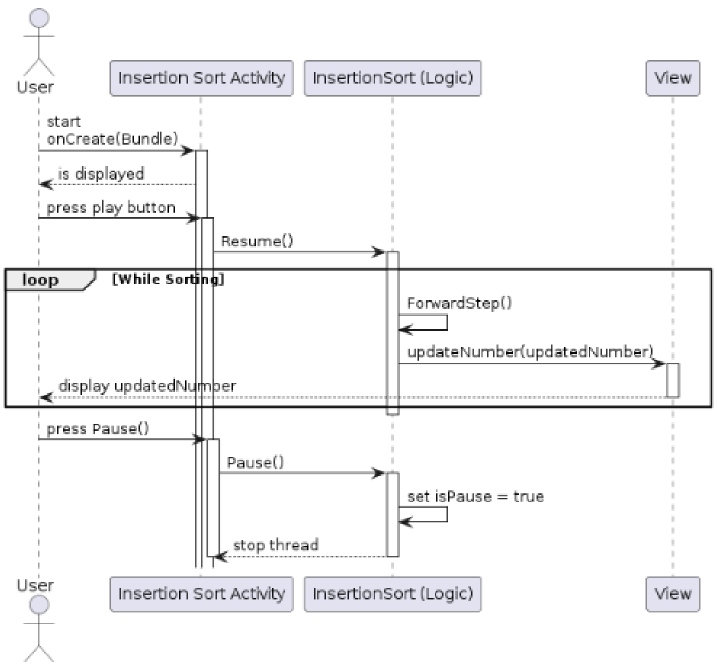
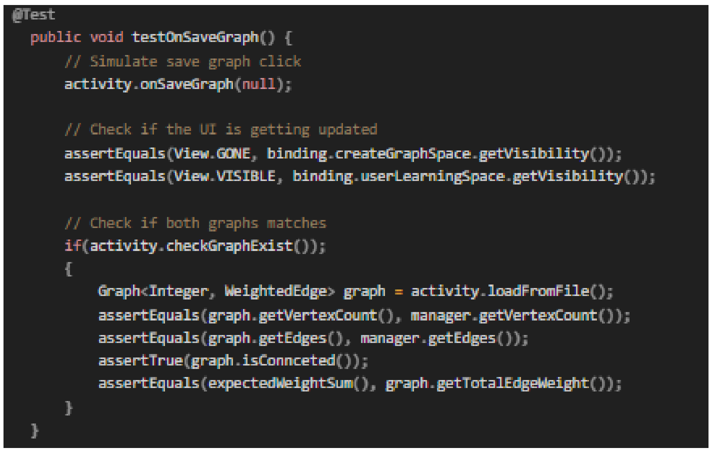


Fig. 21. Shows a sequence diagram of the Insertion Sort Activity.

The project's implementation of a comprehensive suite of unit tests facilitates subsequent integration testing. By thoroughly testing each small component individually, we ensure that these components function correctly in isolation. This preliminary testing phase is crucial as it allows for a smoother transition to integration testing, where these components are assembled to verify the overall system functionality. Thus, integration testing becomes significantly more manageable and effective, enabling us to assess whether the integrated components work synergistically as intended. Figure 22 illustrates the functionality of an integration test for the 'save graph' feature. This function is triggered to verify whether the graph is in the correct format before saving it to a predetermined path. Subsequently, the system checks whether the user interface (UI) updates as expected. Following this, both the saved graph and its prior version are retrieved for comparison to ascertain consistency between the two. The test is deemed successful if the saved graph matches the pre-saved version and fails if the graph does not load correctly, which aligns with the intended outcome of the test.



```java
@Test
  public void testOnSaveGraph() {
      // Simulate save graph click
      activity.onSaveGraph(null);

      // Check if the UI is getting updated
      assertEquals(View.GONE, binding.createGraphSpace.getVisibility());
      assertEquals(View.VISIBLE, binding.userLearningSpace.getVisibility());

      // Check if both graphs matches
      if(activity.checkGraphExist());
      {
          Graph<Integer, WeightedEdge> graph = activity.loadFromFile();
          assertEquals(graph.getVertexCount(), manager.getVertexCount());
          assertEquals(graph.getEdges(), manager.getEdges());
          assertTrue(graph.isConncected());
          assertEquals(expectedWeightSum(), graph.getTotalEdgeWeight());
      }
  }
```

Fig. 22. Shows an example of integration testing.

5.3 System Testing

System testing is a comprehensive testing phase in which the complete and fully integrated software product is examined. The objective is to evaluate the system's compliance with the specified requirements and ensure it is ready for deployment.

The system was tested at multiple stages, with manual testing conducted after each development phase, especially following the implementation of new logic and interactions, where we enabled immediate identification and resolution of functional errors and issues with component interactions. Upon completing all pages and modules, a manual examination was taken, which entailed an exhaustive walkthrough of each page, carefully verifying every functionality and input field. Such a detailed inspection ensured that all aspects of the system were functional and according to application requirements.

5.4 User-Base Testing

User-based testing is a crucial phase where the software is evaluated from the end user's perspective to ensure it meets their needs and expectations. It involves real-world scenarios to validate the system's functionality, usability, and reliability under typical usage conditions.

Following the final implementation of the system and before the full deployment, the application was distributed to users in a pre-release form. Given that the system was developed for the Android platform, an APK (Android Package Kit) file, which is essentially an executable file for Android devices, was utilized for this purpose. The APK enabled users to install and interact with the application on their devices without officially publishing it on the Google Play Store.

After that, users were invited to provide feedback on their experience through a Google Form to elicit comprehensive evaluations of the system’s performance, usability, and overall satisfaction. The insights offer a glimpse into genuine user interactions and identify key areas for improvement. While the user interface and interactions, particularly with the graphical elements, were widely commended for their aesthetic appeal and functionality, respondents expressed concerns regarding the accessibility features of the app. Notably, the lack of support for dark themes, color blindness accommodations, and interactive sound feedback restricts usability for users with visual impairments. Additionally, the inability to zoom in and out was pointed out as a limitation that could affect user engagement. These feedback points are critical as they highlight the essential enhancements needed to make the application more inclusive and user-friendly before its full-scale launch. Moreover, the feedback underscored the need for a broader variety of algorithms and more precise categorization within the system, aligning with the earlier observations that such improvements are necessary to fully optimize the application's educational potential.

Chapter 6: Conclusion

6.1 Challenges and Solutions

Initially, the primary optical was getting familiarized with Android development. This steep learning curve required diligent study of videos, documentation, and tutorials. To bridge our knowledge gap, we constructed a mock application as a sandbox for understanding UI implementation and interaction within the Android ecosystem.

Furthermore, the project demanded proficiency in custom drawing, as our requirements extended beyond the available pre-designed components. This entailed developing from scratch, including the definition of points, colors, and textures, to achieve the desired visual outcomes. Another significant challenge revolved around providing immediate UI feedback, a complex task within the Android framework due to the necessity of completing all function executions before the **onDraw** method could refresh the UI. To circumvent this, we innovated with the **postDelay** function, strategically delaying subsequent actions until after **onDraw** was invoked, thus maintaining the system's interactivity.

6.2 Future Work and Enhancements

Looking ahead, our application is set to introduce several key enhancements to enrich the user experience and educational value:

- **Personalized User Accounts**: Future updates will allow users to create their own accounts, enabling the system to track individual answers and activities. This feature aims to provide personalized feedback on users' performance, highlighting areas of strength and opportunities for improvement.
- **Algorithm Categorization**: The app will introduce dedicated sections for different types of algorithms, such as a 'Graphs' section for Dijkstra and Depth-First Search (DFS) and an 'Arrays' section for relevant algorithms. This organizational change is designed to improve the learning material's ease of navigation and coherence. In Addition, adding a range of algorithms will offer users a broader spectrum of learning and challenge opportunities.
- **Integration with Educational Institutions**: We plan to facilitate a unique feature where teachers can create specialized accounts to enroll students and tailor algorithm challenges according to their teaching needs. Students will access these custom challenges through an intuitive interface, possibly involving QR code scanning for immediate engagement. This integration aims to create a vibrant educational environment where teachers can directly steer and track their students' learning progress.

6.3 Conclusion

the project presented a challenging yet immensely rewarding journey. The development and refinement of the application not only met most of the initial requirements but also opened avenues for future enhancements based on thorough evaluation. The process of bringing this application to fruition has been a profound learning experience, encompassing a broad spectrum of skills. The project has been a crucible for professional growth, from mastering system design through the adept use of various UML diagrams and figures to navigating the complexities of implementation architecture and rigorous testing protocols. It has furnished us with invaluable insights into the intricate dynamics of software development, underscoring the importance of adaptability, precise planning, and the continuous pursuit of excellence.

References

Bhargava, A. (2016). Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People. Manning Publications.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley.

Sahami, M., Roach, S., Cuadros-Vargas, E., & LeBlanc, R. J. (2013). ACM/IEEE-CS computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Association for Computing Machinery and IEEE Computer Society.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? Communications of the ACM, 56(8), 34-36.

Fincher, S., & Petre, M. (2004). Computer Science Education Research. Routledge.

Simon, B., Anderson, R., Hoyer, C., & Su, J. (2006). Preliminary Experiences with a Tablet PC Based System to Support Active Learning in Computer Science Courses. ITiCSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education.

Naps, T., et al. (2002). Exploring the role of visualization and engagement in computer science education. SIGCSE Bulletin, 35(2), 131-152.

Hansen, S., Narayanan, N.H. and Hegarty, M., 2002. Designing educationally effective algorithm visualizations.

Guo, P. J. (2013). Online Python Tutor: Embeddable Web-based Program Visualization for CS Education. Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE 2013).

Grissom, S., McNally, M., & Naps, T. (2003). Algorithm visualization in CS education: Comparing levels of student engagement. Proceedings of the 2003 ACM Symposium on Software Visualization.

Piaget, J. (1970). Science of Education and the Psychology of the Child. Orion Press.

Lahti, L., Seitamaa-Hakkarainen, P., & Hakkarainen, K. (2014). Flipping the classroom to stir algorithmic thinking: Towards creative learning in CS education. Proceedings of the 9th Workshop in Primary and Secondary Computing Education.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. Journal of Visual Languages & Computing, 13(3), 259-290.

Stasko, J., Domingue, J., Brown, M. H., & Price, B. A. (1993). Software visualization: Programming as a multimedia experience. MIT Press.

Urquiza-Fuentes, J., & Velázquez-Iturbide, J. Á. (2009). A survey of successful evaluations of program visualization and algorithm animation systems. ACM Transactions on Computing Education (TOCE), 9(2), Article 9.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/

Tversky, B., Morrison, J. B., & Betrancourt, M. (2002). Animation: can it facilitate? International Journal of Human-Computer Studies, 57(4), 247-262.

Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2000). The Java Language Specification. Addison-Wesley.

Appendices

A Self-appraisal

A.1 Critical self-evaluation and Personal reflection

Throughout the course of my project, I embarked on a journey that not only enhanced my technical skillset but also refined my professional objectives and personal growth. This critical self-evaluation aims to dissect the various aspects of my learning experience, integrating personal reflections with an analysis of lessons learned along the way.

A significant portion of my project was dedicated to mastering a range of technologies, pivotal for both the project's success and my professional development. I acquired proficiency in Java, delving into the most common design strategies that enhanced my programming capabilities. The learning curve was steep, yet it was instrumental in laying a solid foundation for my coding skills.

Additionally, I familiarized myself with essential tools that streamlined my project design process. Utilizing Figma, I was able to conceptualize and iterate design solutions effectively. PlantUML proved invaluable for drawing complex diagrams using code, which significantly optimized the documentation and planning phases of the project.

The Android framework and environment were new territories that I explored with enthusiasm, driven by a long-standing desire to develop Android applications. This exploration was not merely a technical challenge but a step towards aligning my academic pursuits with my career aspirations in mobile app development. The project provided a practical context to apply what I had learned during my placement year at Feral Interactive, confirming my interest and dedication to this professional path.

Time management emerged as a crucial skill throughout this project. Balancing intensive project work with simultaneous coursework demands required a strategic approach to prioritize tasks effectively. For example, if coursework required programming, I would strategically combine it with some design or documentation aspects of my project. This approach allowed me to manage the programming load effectively without becoming overwhelmed, ensuring that both my coursework and project progressed smoothly, in which the experience has taught me the importance of flexibility and resilience in managing competing deadlines, a skill that will undoubtedly benefit me in future professional settings.

Engaging with academic research was another facet of my project, enriching my understanding and application of theoretical concepts. This skill was not only essential for the academic rigor of the project but also enhanced my ability to synthesize information and apply it in a practical context.

In conclusion, I regard this project as a significant personal achievement. The entire process was not only educational but also deeply satisfying, as it allowed me to meet the established aims and objectives effectively. The obstacles I encountered and subsequently overcame provided me with a comprehensive experience in diverse fields such as app development, user interface design, and performance optimization. Reflecting on the journey, I am pleased with the final outcome and the robust skill set I have developed, which are a testament to the hard work and dedication invested in this project.

A.2 Legal, social, ethical, and professional issues

A.2.1 Legal Issues

The legal framework surrounding the project was considered to ensure compliance with copyright laws and licensing regulations. A central concern was the preservation of rights concerning the software and tools utilized throughout the development process. In this regard, all intellectual property rights related to the project's output were upheld, ensuring that all contributions and developments remained properly attributed and legally protected.

A notable aspect of the project's legal considerations involved the use of various software tools and libraries, which were selected based on their licensing terms to avoid any infringement issues. The JUNG library, which was used for generating visual representations and handling network data, is a notable example. This library is free and open-source, offering a compliant solution for integration into the project without the complications of license restrictions.

Similarly, other essential tools such as Figma and Android Studio, along with its related plugins, were utilized under their respective free licenses. These tools provided robust support for design and development tasks, ensuring that the project adhered to legal standards while enabling a high level of functionality and user experience design.

Through careful selection and use of software tools and libraries, the project maintained a clear legal standing, ensuring that all utilized resources were in alignment with copyright and licensing norms.

A.2.2 Social Issues

At present, my application presents minimal social issues; however, there is a notable concern regarding accessibility. The application does not support Android devices running on versions older than Android 29, which could restrict access for users with older technology. Additionally, the app currently lacks several essential features that would make it more accessible to users with disabilities. These include screen reader compatibility, text-to-speech functions, and adjustable font sizes. Addressing these shortcomings is crucial to ensure that the app is inclusive and accessible to a broader audience, thereby enhancing user experience and adherence to best practices in software development.

A.2.3 Ethical Issues

No ethical concerns are linked to this project, as it does not present any direct ethical implications.

A 2.4 Professional Issues

In developing AlgoLearn, I have diligently adhered to professional standards to ensure the application operates effectively and provides accurate educational content. Extensive testing was conducted to verify that all functionalities, particularly the algorithms, perform as intended without errors. This quality assurance process involved various tests designed to evaluate the app’s stability, usability, and educational effectiveness.

To further enhance the app's quality and relevance, I sought feedback from actual users through structured mechanisms, such as Google Forms. This approach allowed me to gather valuable insights and suggestions from diverse users, including potential students and educators. This feedback played a crucial role in refining the application, ensuring it met the initial educational goals while closely matching user expectations and learning requirements.

Through these measures, I have maintained high professionalism in the app's development, demonstrating a commitment to quality, accuracy, and user-centered design.

B Software Initial requirements

Figure 23 presents the application requirement table, clearly illustrating that most of the desired features have been successfully implemented, except for deployment to the Google Play Store. This implementation record reflects positively on the substantial progress and achievements made in the development of the application.

1: Height priority.

2: Medium priority.

3: Low propriety.

Feature	**Priority**	**Implemented (Yes/No)**
3 Algorithm implementation	1	Yes
Well-designed and Responsive User Interface	1	Yes
Interactive Algorithm Learning Experience	1	Yes
Testing implementation	1	Yes
Deployment to Google Play Store	2	No
User feedback collection	3	Yes
Legal and Compliance Checks	3	Yes

Fig. 23. Requirement table

C Development Tools

Android Studio: main IDE, can be downloaded on:

https://developer.android.com/studio

JUNG Library: main website:

https://jung.sourceforge.net

GitHub: for hosting the application repository.

https://github.com/uol-feps-soc-comp3931-2324-classroom/final-year-project-JustAbdou

PlantUML: for drawing diagrams using code.

https://plantuml.com/

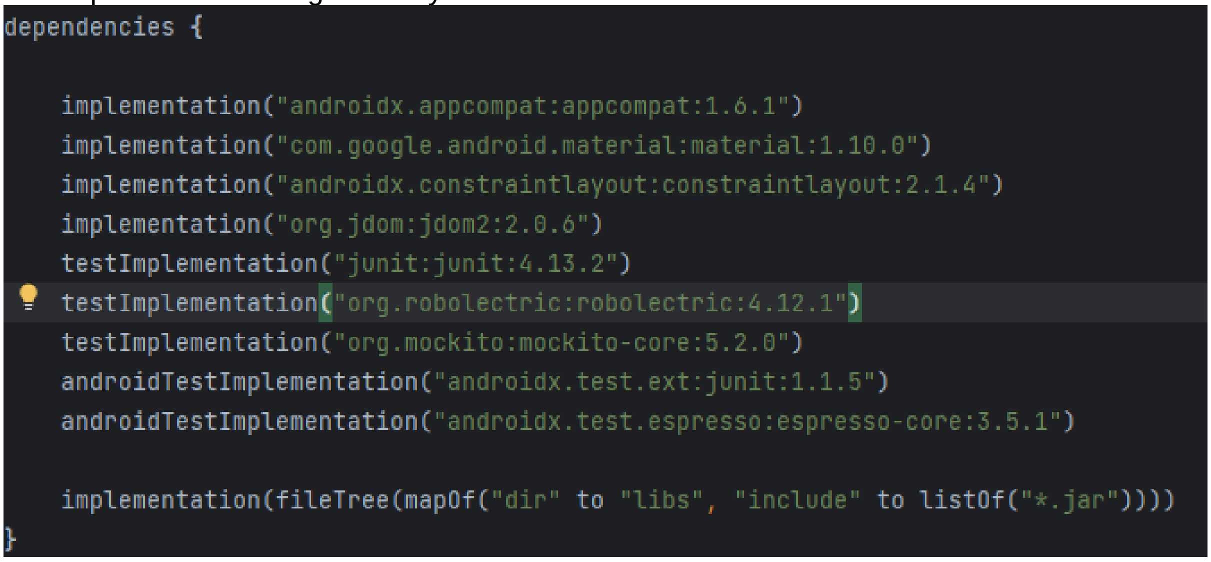
Gradle: java build system.

https://gradle.org/

Figma: for project prototype.

https://www.figma.com

Additionally, Figure 24 displays the comprehensive set of libraries that facilitated the development and testing of the system.

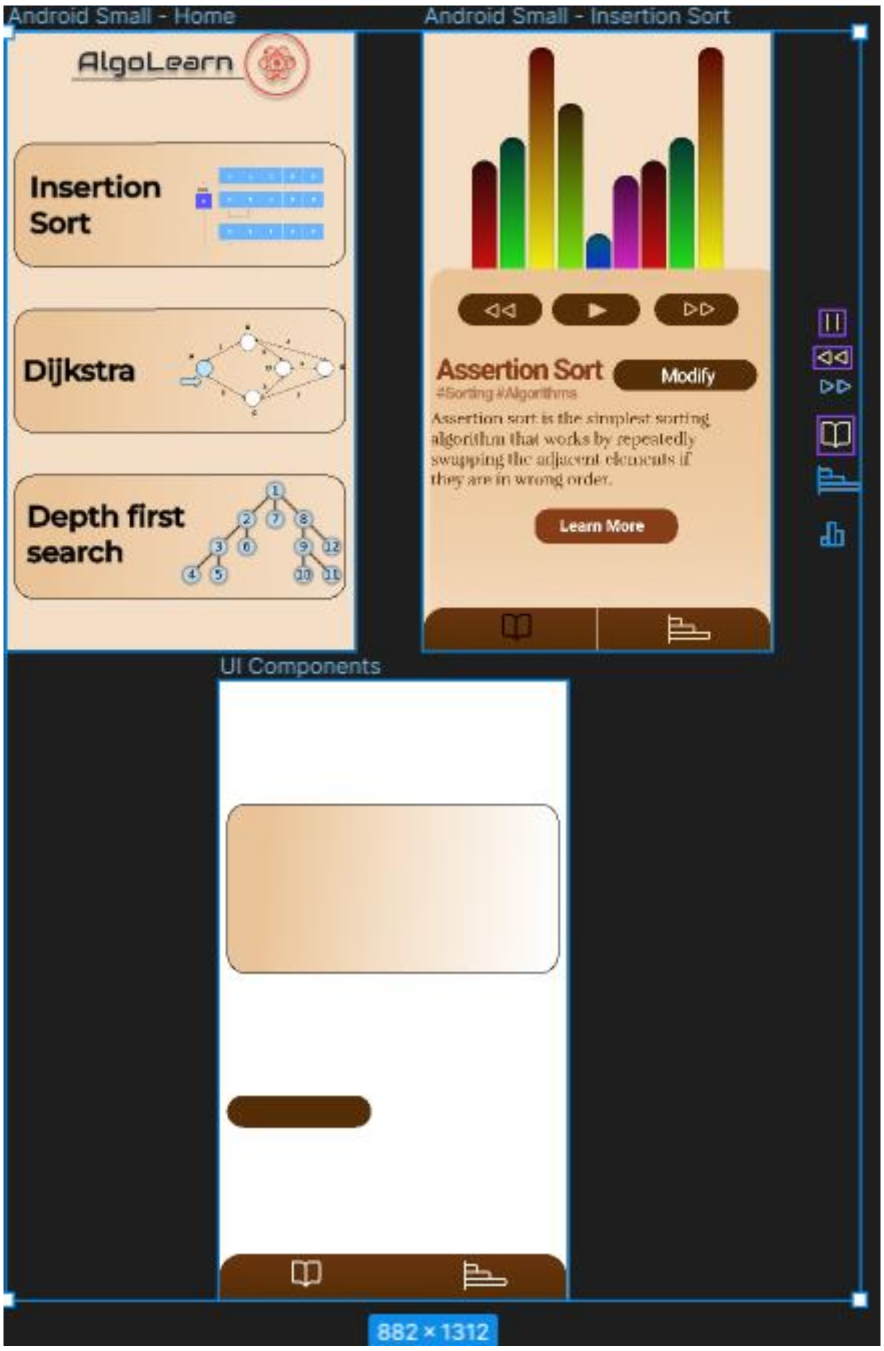


```
dependencies {

    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.10.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation("org.jdom:jdom2:2.0.6")
    testImplementation("junit:junit:4.13.2")
    testImplementation("org.robolectric:robolectric:4.12.1")
    testImplementation("org.mockito:mockito-core:5.2.0")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")

    implementation(fileTree(mapOf("dir" to "libs", "include" to listOf("*.jar"))))
}
```

Fig. 24. Shows all the dependencies used on the project.

D Application Prototype

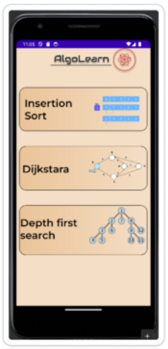
Figure 25 illustrates the application prototype, showcasing the main page and the insertion sort interface. All other pages maintain a consistent structure and utilize the same components. Additionally, there is a dedicated framework designed to house these components, allowing for their reuse in future developments.

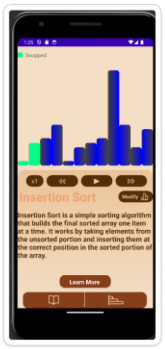


Assertion sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

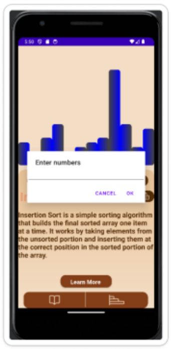
Figure. 25. Shows a prototype of the application

E Application Overview

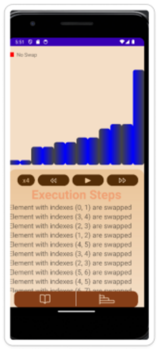




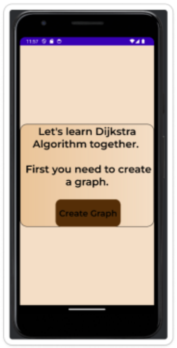
Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It works by taking elements from the unsorted portion and inserting them at the correct position in the sorted portion of the array.



Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It works by taking elements from the unsorted portion and inserting them at the correct position in the sorted portion of the array.

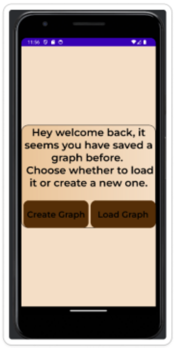


```
Element with indexes (0, 1) are swapped
Element with indexes (3, 4) are swapped
Element with indexes (2, 3) are swapped
Element with indexes (1, 2) are swapped
Element with indexes (4, 5) are swapped
Element with indexes (3, 4) are swapped
Element with indexes (2, 3) are swapped
Element with indexes (5, 6) are swapped
Element with indexes (4, 5) are swapped
```

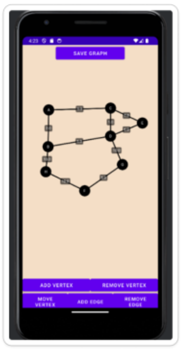


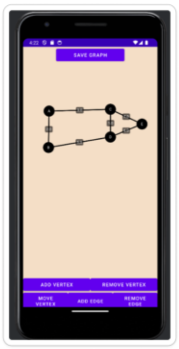
Let's learn Dijkstra Algorithm together.

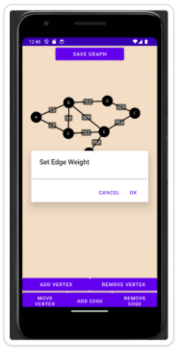
First you need to create a graph.

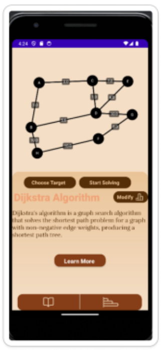


Hey welcome back, it seems you have saved a graph before. Choose whether to load it or create a new one.

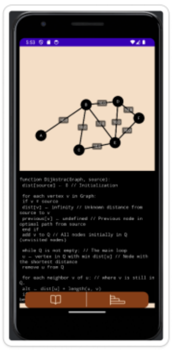




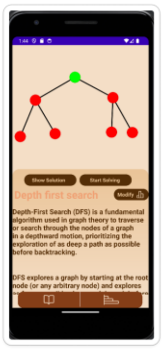




Dijkstra's algorithm is a graph search algorithm that solves the shortest path problem for a graph with non-negative edge weights, producing a shortest path tree.

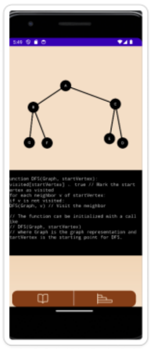


```
function Dijkstra(Graph, source):
 dist[source] ← 0 // Initialization

 for each vertex v in Graph:
 if v ≠ source
 dist[v] ← infinity // Unknown distance from source to v
 previous[v] ← undefined // Previous node in optimal path from source
 end if
 add v to Q // All nodes initially in Q (unvisited nodes)

 while Q is not empty: // The main loop
 u ← vertex in Q with min dist[u] // Node with the shortest distance
 remove u from Q

 for each neighbor v of u: // where v is still in Q.
 alt ← dist[u] + length(u, v)
```



Depth-First Search (DFS) is a fundamental algorithm used in graph theory to traverse or search through the nodes of a graph in a depthward motion, prioritizing the exploration of as deep a path as possible before backtracking.

DFS explores a graph by starting at the root node (or any arbitrary node) and explores



```
unction DFS(Graph, startVertex):
visited[startVertex] ← true // Mark the start
ertex as visited
for each neighbor v of startVertex:
if v is not visited:
DFS(Graph, v) // Visit the neighbor

// The function can be initialized with a call
ike
// DFS(Graph, startVertex)
// where Graph is the graph representation and
tartVertex is the starting point for DFS.
```

F Time Management

Time management presented the most challenging aspect of this project, as I had to juggle responsibilities between work and coursework from other modules. Despite my difficulties and struggles, I adhered closely to the planned schedule. Figure 26 displays a Gantt chart that outlines the project's progress, illustrating how the various phases were navigated over time.

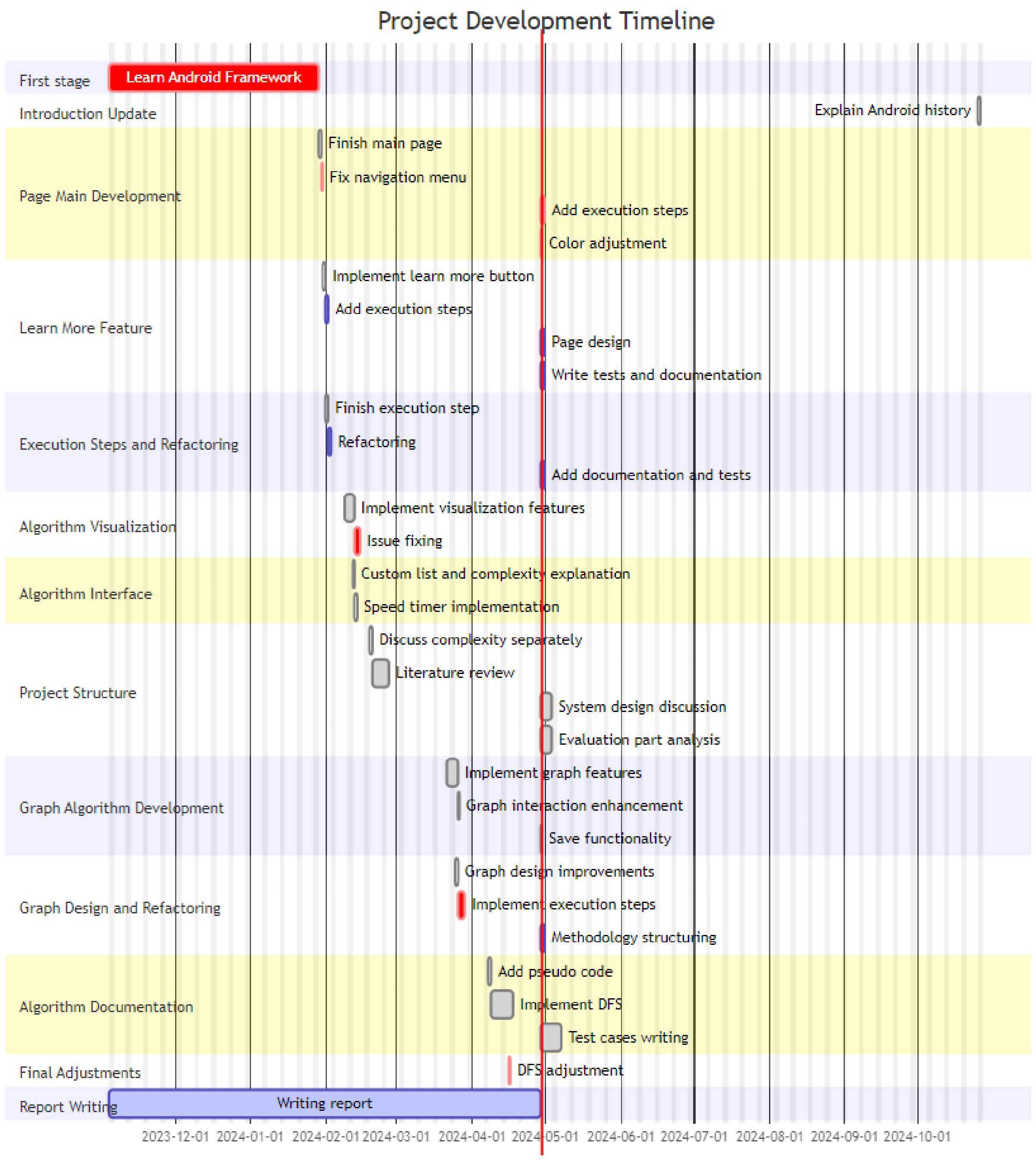
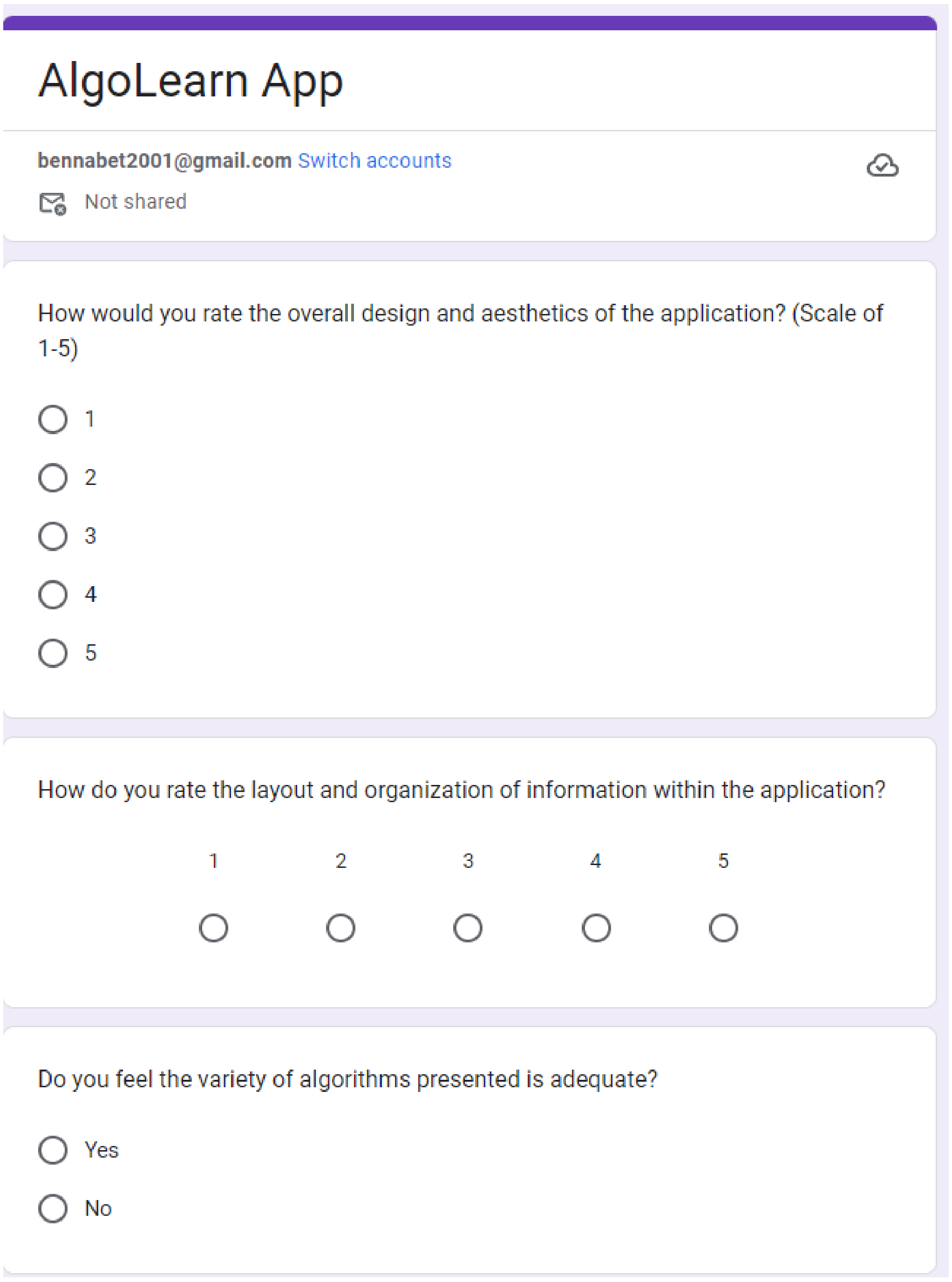


Fig. 26. Time Management Gantt Chart

G User Feedback forms



AlgoLearn App

bennabet2001@gmail.com Switch accounts

Not shared

How would you rate the overall design and aesthetics of the application? (Scale of 1-5)

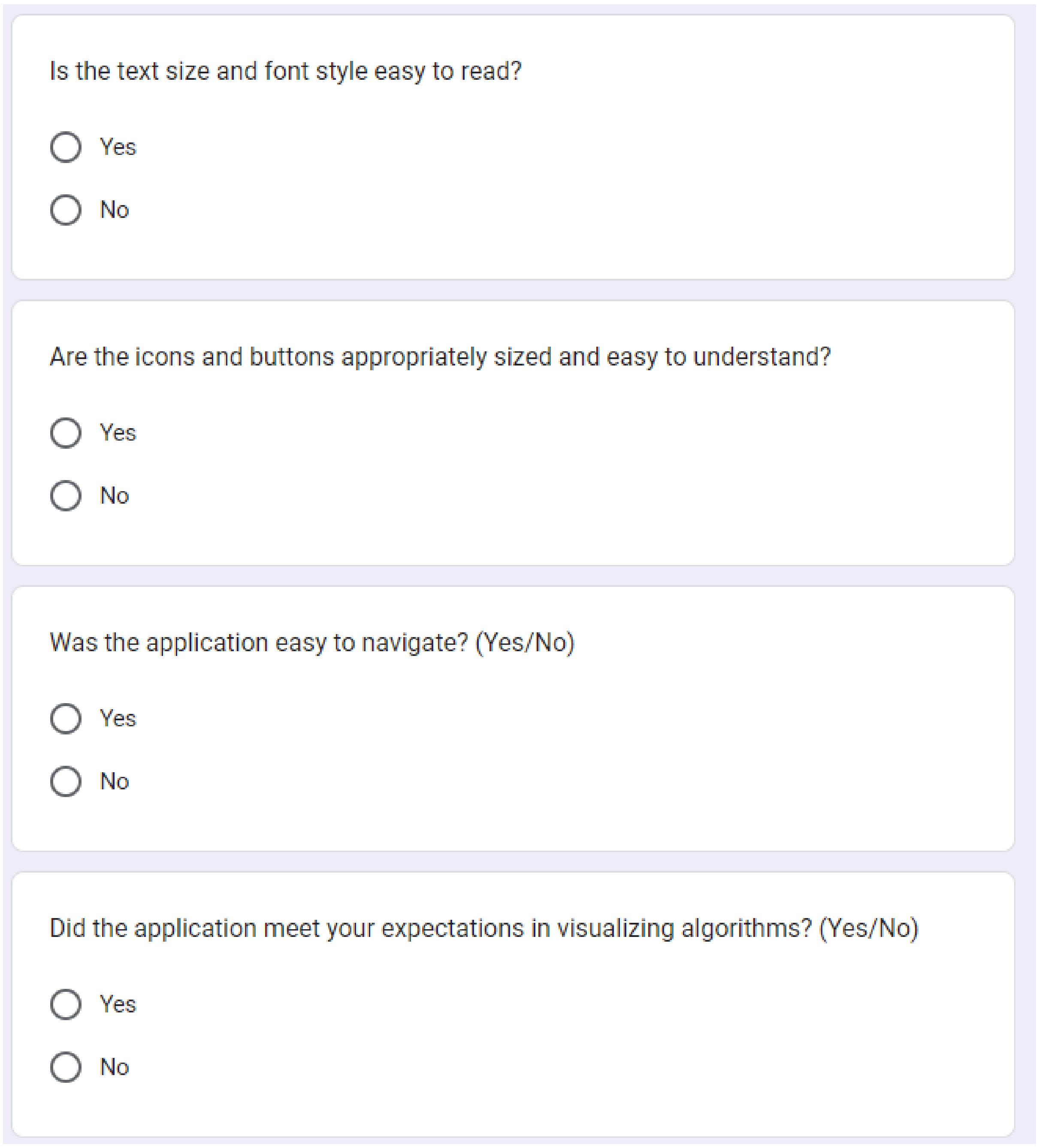
- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

How do you rate the layout and organization of information within the application?

1	2	3	4	5
○	○	○	○	○

Do you feel the variety of algorithms presented is adequate?

- ○ Yes
- ○ No



Is the text size and font style easy to read?

○ Yes

○ No

Are the icons and buttons appropriately sized and easy to understand?

○ Yes

○ No

Was the application easy to navigate? (Yes/No)

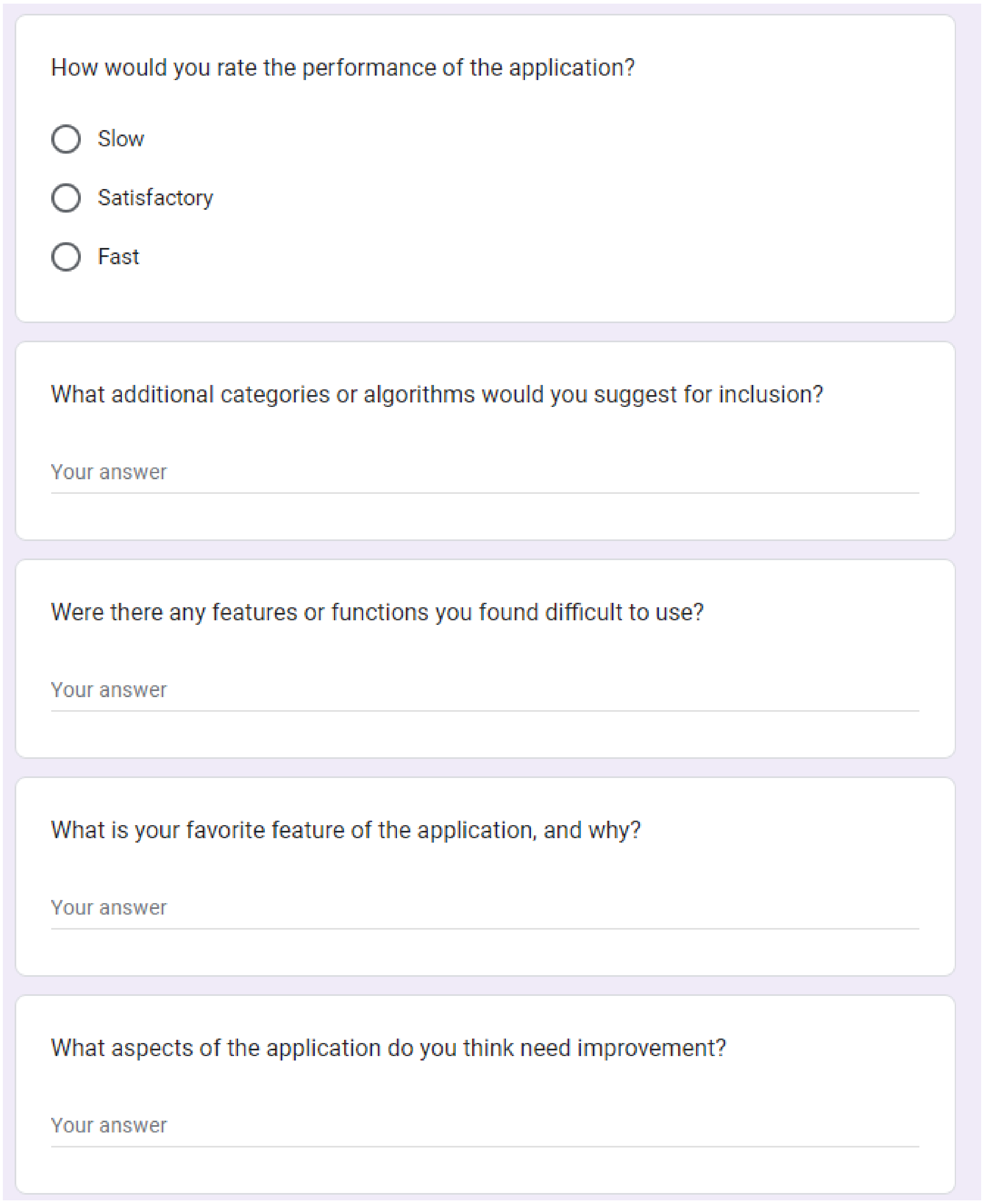
○ Yes

○ No

Did the application meet your expectations in visualizing algorithms? (Yes/No)

○ Yes

○ No


How would you rate the performance of the application?

- Slow
- Satisfactory
- Fast

What additional categories or algorithms would you suggest for inclusion?

Your answer

Were there any features or functions you found difficult to use?

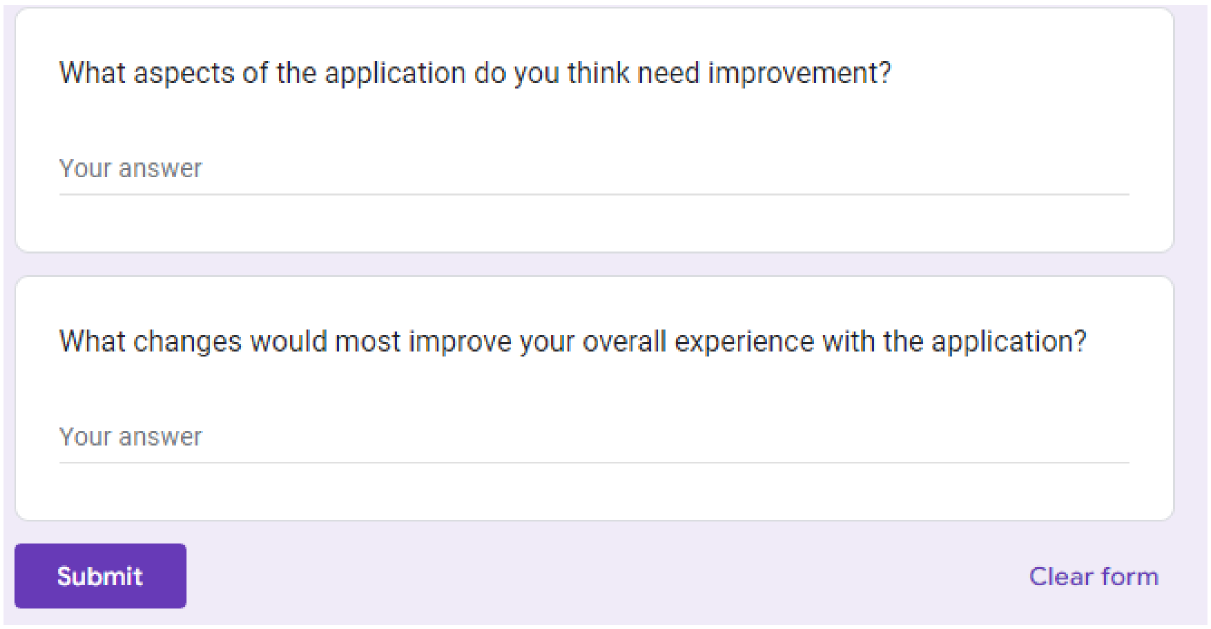
Your answer

What is your favorite feature of the application, and why?

Your answer

What aspects of the application do you think need improvement?

Your answer


What aspects of the application do you think need improvement?

Your answer

What changes would most improve your overall experience with the application?

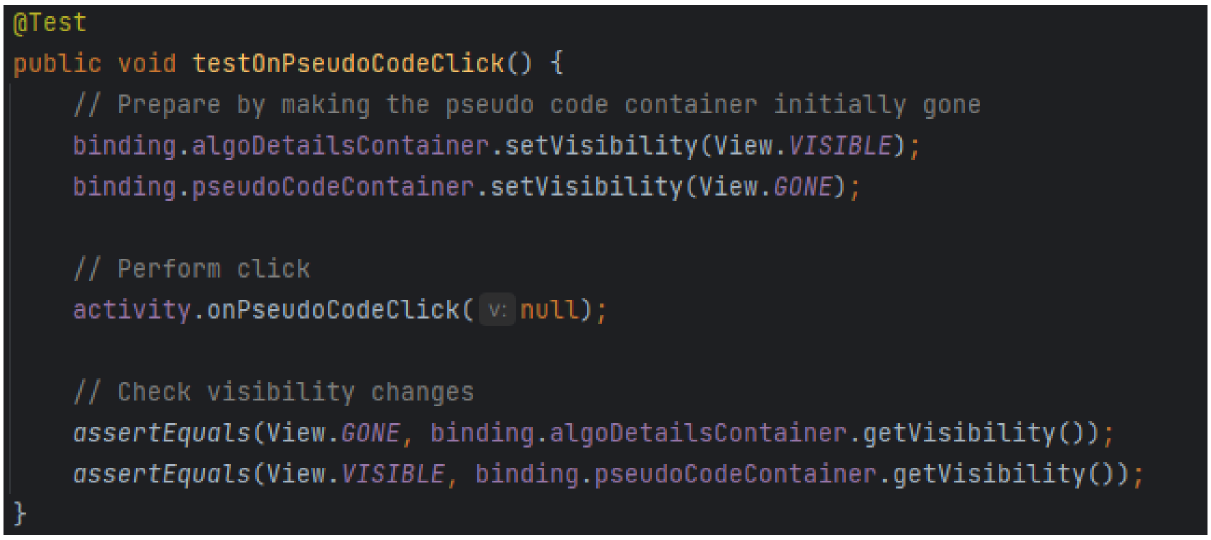
Your answer

H Testing

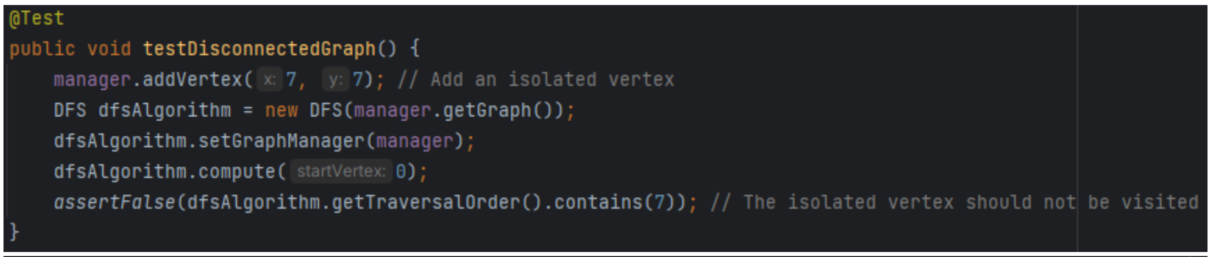
The comprehensive testing framework employed in this project ensures the application's robustness and reliability. Initially, a series of unit tests was meticulously crafted to evaluate each individual component of the software, verifying the correct functionality of isolated units in various scenarios. This foundational phase of testing confirmed the integrity of the smallest elements of our application, setting a solid base for subsequent integration testing.

Integration tests were methodically conducted following the unit testing to ascertain the seamless interaction between these previously tested units. This phase focused on the cohesive operation of multiple components working together, simulating real-world usage to ensure that the integrated system met all functional requirements. Specifically, the integration tests covered data saving and retrieval scenarios, UI responsiveness, and error handling, effectively validating the overall system's performance.

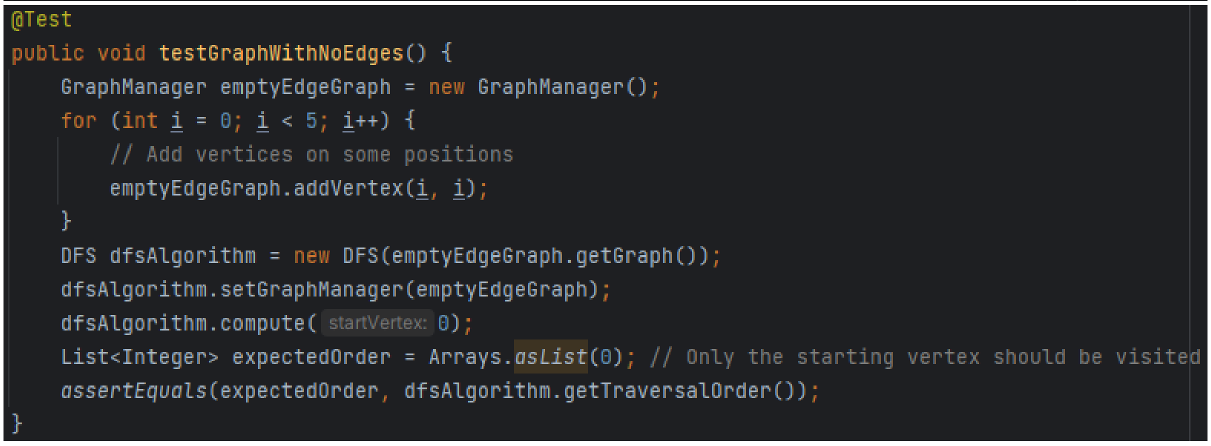
The combined unit and integration testing approach has thoroughly vetted the application across all critical aspects. Every test case was designed to mirror potential real-life application scenarios, ensuring that the application performs optimally under test conditions and maintains this standard in live environments. The successful execution of these tests across various application architecture layers provides strong evidence that the software is functioning as intended, bolstering confidence in its reliability and readiness for deployment. The following diagrams shows a sample proof the conducted tests.



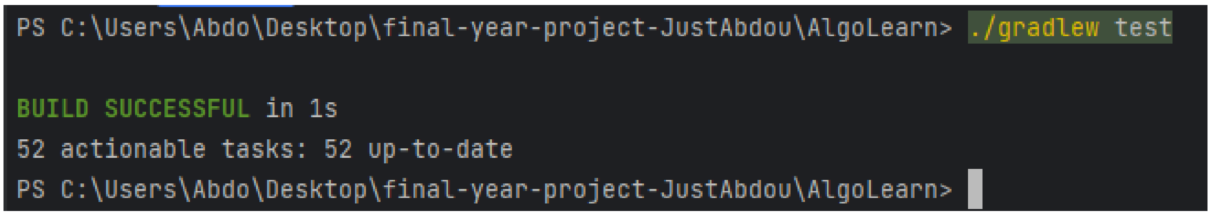
```java
@Test
public void testOnPseudoCodeClick() {
    // Prepare by making the pseudo code container initially gone
    binding.algoDetailsContainer.setVisibility(View.VISIBLE);
    binding.pseudoCodeContainer.setVisibility(View.GONE);

    // Perform click
    activity.onPseudoCodeClick( v: null);

    // Check visibility changes
    assertEquals(View.GONE, binding.algoDetailsContainer.getVisibility());
    assertEquals(View.VISIBLE, binding.pseudoCodeContainer.getVisibility());
}
```



```java
@Test
public void testDisconnectedGraph() {
    manager.addVertex( x: 7,  y: 7); // Add an isolated vertex
    DFS dfsAlgorithm = new DFS(manager.getGraph());
    dfsAlgorithm.setGraphManager(manager);
    dfsAlgorithm.compute( startVertex: 0);
    assertFalse(dfsAlgorithm.getTraversalOrder().contains(7)); // The isolated vertex should not be visited
}
```



```java
@Test
public void testGraphWithNoEdges() {
    GraphManager emptyEdgeGraph = new GraphManager();
    for (int i = 0; i < 5; i++) {
        // Add vertices on some positions
        emptyEdgeGraph.addVertex(i, i);
    }
    DFS dfsAlgorithm = new DFS(emptyEdgeGraph.getGraph());
    dfsAlgorithm.setGraphManager(emptyEdgeGraph);
    dfsAlgorithm.compute( startVertex: 0);
    List<Integer> expectedOrder = Arrays.asList(0); // Only the starting vertex should be visited
    assertEquals(expectedOrder, dfsAlgorithm.getTraversalOrder());
}
```



```
PS C:\Users\Abdo\Desktop\final-year-project-JustAbdou\AlgoLearn> ./gradlew test

BUILD SUCCESSFUL in 1s
52 actionable tasks: 52 up-to-date
PS C:\Users\Abdo\Desktop\final-year-project-JustAbdou\AlgoLearn>
```

I Activity Life Cycle

The lifecycle of an Android activity, as depicted in the provided diagram, is a fundamental concept in developing Android applications. It encompasses several states that an activity transitions through, each corresponding to a specific callback method that allows developers to manage the application's behavior as it interacts with the user and the system. Understanding and effectively managing these lifecycle stages is crucial for creating robust, efficient, and user-friendly applications.

onCreate(): This initial lifecycle callback is invoked when the activity is first created. Developers must perform essential initializations here, such as setting up the user interface and initializing data structures. Following onCreate(), the lifecycle progresses to onStart().

onStart(): In this phase, the activity becomes visible to the user. This callback prepares the activity to enter the foreground and become interactive, setting the stage for onResume().

onResume(): When the activity enters the foreground, onResume() is triggered, signifying that the activity is ready to interact with the user. This state indicates that the activity has the highest priority and will receive all user inputs.

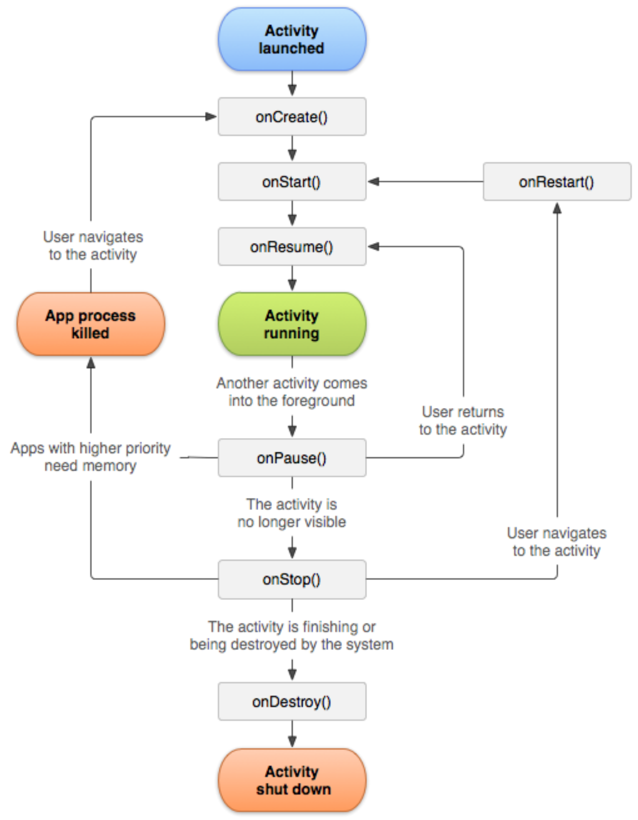
onPause(): This method is called as the first signal of the user transitioning away from the activity, such as by navigating to a new activity or receiving an incoming call. Activities should pause ongoing operations that should not continue while in the background, like pausing a game or video playback.

onStop(): Triggered when the activity is no longer visible, onStop() is used to suspend or terminate operations that are not necessary when the activity is not in the foreground. It is also an appropriate time to release or adjust resources that are limited or costly in nature.

onRestart(): This callback is executed if the activity is coming back to the foreground from the stopped state, signifying a restart of interrupted processes that were halted in onStop().

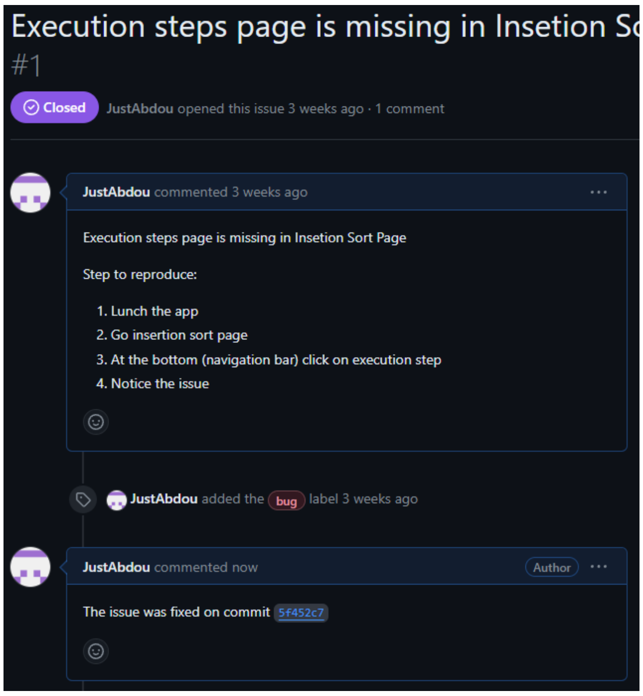
onDestroy(): Serving as the final phase in the lifecycle, onDestroy() is invoked when the activity is finishing, either due to a call to finish() or as a result of the system temporarily destroying the instance for resource optimization.

Activity Shutdown: This state occurs when the activity is completely terminated, either by user action or system conditions, and is removed from memory.



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

J Git



Execution steps page is missing in Insetion Sc

#1

Closed JustAbdou opened this issue 3 weeks ago · 1 comment

JustAbdou commented 3 weeks ago

Execution steps page is missing in Insetion Sort Page

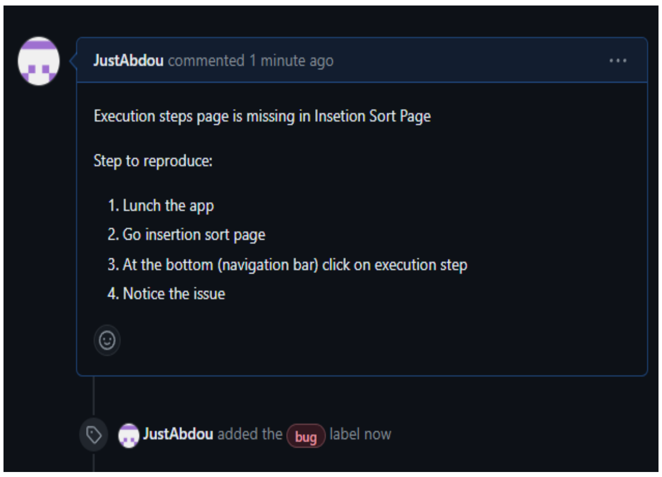
Step to reproduce:

1. Lunch the app
2. Go insertion sort page
3. At the bottom (navigation bar) click on execution step
4. Notice the issue

JustAbdou added the bug label 3 weeks ago

JustAbdou commented now Author

The issue was fixed on commit 5f452c7



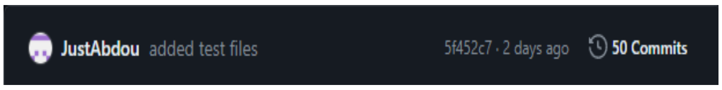
JustAbdou commented 1 minute ago

Execution steps page is missing in Insetion Sort Page

Step to reproduce:

1. Lunch the app
2. Go insertion sort page
3. At the bottom (navigation bar) click on execution step
4. Notice the issue

JustAbdou added the bug label now



JustAbdou added test files 5f452c7 · 2 days ago 50 Commits