A

Project Report

On

BruhDive – An Online Quiz Platform

Submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Computer Applications (BCA)

of

Kavikulguru Kalidas Sanskrit University

Submitted by

Poonam Prithviraj Lohar

Under the guidance of

<u>Prof. Sneha Shashikant Lokhande</u>

Kavikulguru Kalidas Sanskrit University’s

Bakliwal Foundation Collage of Arts, Commerce & Science

Vashi.

BATCH: 2022-25

Kavikulguru Kalidas Sanskrit University's

Bakliwal Foundation Collage of Arts, Commerce & Science

Vashi.

CERTIFICATE

This is to certify that the project entitled **BruhDive – An Online Quiz Platform** undertaken at the PCP Center: Bakliwal Foundation of Arts, Commerce & Science, Vashi, New Mumbai by **MISS POONAM PRITHVIRAJ LOHAR** holding **Seat No. (PRN : 2022018100102235)** Studying **Bachelor of Computer Applications** Semester – VI has been satisfactorily completed as prescribed by the Kavikulguru Kalidas Sanskrit University, during the year 2024 – 2025.

Project In-charge **Co-Ordinator**

External Examiner

Internal Examiner **Principal**

DECLARATION

I hereby declare that the project entitled, “BruhDive – An Online Quiz Platform” done at Bakliwal Foundation College of Arts, Commerce and Science (BFCACS), has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of BACHELOR OF COMPUTER APPLICATION (BCA) to be submitted as final semester project as part of our curriculum.

Poonam Prithviraj Lohar

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude and appreciation to all those who have contributed to the successful completion of my BCA project, BruhDive – An Online Quiz Platform. Their continuous support, encouragement, and guidance have been instrumental in bringing this project to life.

I extend my sincere thanks to my project guides, Coordinator H.O.D. Prof. Sneha Shashikant Lokhande and Prof. Ankit Srivastava, for their unwavering support, expert guidance, and constructive feedback, which have been a constant source of motivation throughout my project journey.

I would also like to acknowledge the efforts of everyone who directly or indirectly contributed to the successful execution of this project. Their support has played a vital role in shaping this project into its final form.

Lastly, I extend my sincere thanks to Principal Dr. Sharadkumar Shah, H.O.D., Prof. Sneha Shashikant Lokhande, Prof. Shaikh Mohammed Umar, Prof. Divya Patil, Prof. Kalyani Kulkarni, and Prof. Ankit Srivastava for their valuable guidance and encouragement throughout this academic journey.

Thanking you,

Poonam Prithviraj Lohar

TABLE OF CONTENTS

1. ABSTRACT

Aspiring tech learners often face uncertainty when assessing their programming skills and identifying the right career paths. BruhDive – An Online Quiz Platform is a web-based platform designed to help users evaluate their proficiency across multiple programming languages through structured quizzes. With a clean and intuitive interface, the platform makes learning more engaging by providing real-time feedback, level-based performance evaluation, and targeted topic suggestions.

In addition to language-specific quizzes, BruhDive – An Online Quiz Platform features a specialized AI-powered assessment that analyzes a user’s performance across various technical domains and recommends suitable career tracks. The platform also offers curated resources like roadmaps, project ideas, and certifications to support continuous learning. This project demonstrates how guided, skill-focused assessments can empower tech learners to grow confidently and discover personalized learning paths.

1.1. INTRODUCTION

1.1.1. BACKGROUND

In today’s fast-paced tech-driven world, aspiring developers and IT learners often struggle to identify their strengths and weaknesses across various programming domains. With a multitude of learning platforms available, it can be overwhelming to find a tool that is both concise and effective in skill evaluation. *BruhDive* was developed with this gap in mind—a lightweight, browser-based platform designed to allow users to test their technical knowledge across multiple programming languages without any complex sign-up processes.

Unlike traditional learning platforms, *BruhDive* emphasizes a “grab-and-go” model where users can instantly start learning and testing themselves without creating an account. With features like structured quizzes per language, detailed results analysis, and topic-based recommendations, the platform ensures a smooth user experience. Additionally, it includes a special multi-domain quiz that leverages AI to offer diverse questions for broader technical exposure, backed by a fallback system for reliability.

1.1.2. <u>OBJECTIVES</u>

- To build an interactive quiz platform tailored for aspiring tech learners to assess their programming knowledge.
- To provide structured sets of 25 questions per language, categorized by skill levels.
- To help users identify their current proficiency level through real-time scoring and analysis.
- To offer topic-based learning recommendations based on the user's quiz performance.
- To include a multi-domain “Test Yourself” quiz for broader exposure using AI-generated questions.
- To deliver a clean, user-friendly interface that guides users smoothly through the quiz-taking process.
- To provide centralized access to curated tech resources including courses, roadmaps, internships, and project ideas.

1.1.3. <u>PURPOSE, SCOPE AND APPLICABILITY</u>

Purpose

The purpose of *BruhDive* is to offer a practical and engaging platform for tech learners to test and track their progress across various programming languages. The system aims to simplify the learning journey by allowing users to take quizzes, assess their proficiency level, and receive guidance on what topics to explore next. Its no-login model promotes ease of use and fast access, making it ideal for students who want to evaluate themselves quickly and efficiently.

Scope

BruhDive currently supports quizzes in multiple programming languages like Python, JavaScript, Java, PHP, Kotlin, SQL, and more. Each language includes a dedicated set of 25 carefully curated questions. The platform also includes a “Test Yourself” quiz powered by AI to offer questions across domains like Web Development, Cybersecurity, and Data Structures. While it doesn’t store user data in a backend database, it uses local storage to maintain quiz history for the session. The platform also features a well-organized resource section offering external links to internships, courses, roadmaps, events, and more—all curated in one place for convenience.

Applicability

The project primarily serves tech learners—students, freshers, and self-taught developers—seeking quick and structured self-assessment tools. While the AI-based career guidance is a supplementary feature, the core functionality focuses on helping users identify their skill levels and guide them toward suitable learning paths. *BruhDive* can be used by educational institutions, coding clubs, or individuals for self-evaluation and practice. Its modular, frontend-heavy design also makes it easily extendable in the future to support user logins, databases, or integration with learning management systems.

2. LITERATURE SURVEY

- With the growing popularity of self-paced and remote learning, there is a strong demand for platforms that offer interactive and effective ways to test programming skills. Most available platforms either rely heavily on tutorial-based learning or focus on competitive programming, which can be intimidating for beginners. There exists a need for lightweight platforms that provide structured evaluations and learning direction without overwhelming the user.
- Research in educational technology shows that quiz-based learning methods significantly improve concept retention, especially when paired with instant feedback. Learners are more likely to engage when they receive immediate insights about their strengths and weaknesses, enabling more personalized and focused study.
- Many platforms provide learning content, but few offer quiz sets across multiple programming languages, domain-wise testing, and personalized performance feedback all in one place. **BruhDive** attempts to bridge this gap by providing 25-question quizzes for various tech languages, each categorized by difficulty level and skill type.
- Additionally, **BruhDive** provides a separate AI-powered quiz mode that allows users to evaluate their aptitude across different tech domains. This helps users get a better understanding of which tech field might interest them most. Though it doesn't claim to offer full career guidance, it gives helpful insights that many beginners look for.
- As a browser-based platform with no sign-in required, **BruhDive** prioritizes accessibility. Its "grab-and-go" approach makes it ideal for aspiring tech learners who want fast, structured evaluation with topic recommendations—all without dealing with account creation, storage, or setup hassles.

2.1. USER SURVEY FOR REQUIREMENT VALIDATION

To ensure BruhDive aligns with real user needs, a pre-development user survey was conducted. This primary research complements the literature by identifying specific challenges, preferences, and expectations from our target audience: coding learners and aspiring developers.

Survey Methodology

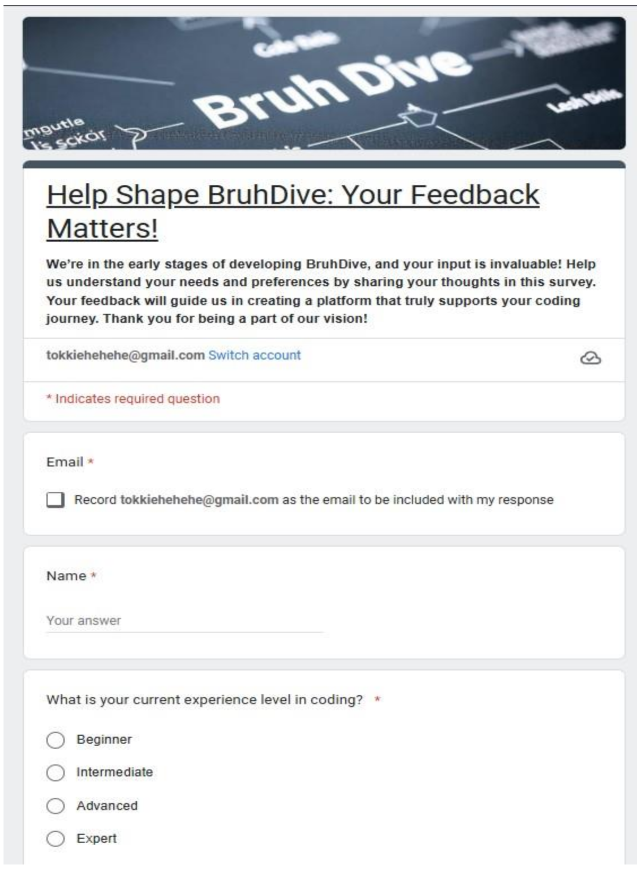
- Objective:
  The goal of the survey was to collect data about user preferences in areas such as quiz features, career guidance, device compatibility, and feedback mechanisms. This data helped in designing a user-centered quiz platform.
- Target Audience:
  The survey focused on students, recent graduates, and beginner-level coders who are actively looking to enhance their technical skills through interactive learning methods.
- Sampling Method:
  Convenience sampling was employed, targeting individuals with a background or interest in coding. This method allowed efficient and relevant data collection within a short time frame.

Methodology

- Sampling Method: Convenience sampling.
- Tool Used: Google Forms.

Questionnaire Design

The survey consisted of 15 key questions, broadly categorized under the following themes:



Help Shape BruhDive: Your Feedback Matters!

We're in the early stages of developing BruhDive, and your input is invaluable! Help us understand your needs and preferences by sharing your thoughts in this survey. Your feedback will guide us in creating a platform that truly supports your coding journey. Thank you for being a part of our vision!

tokkiehehehe@gmail.com Switch account

* Indicates required question

Email *

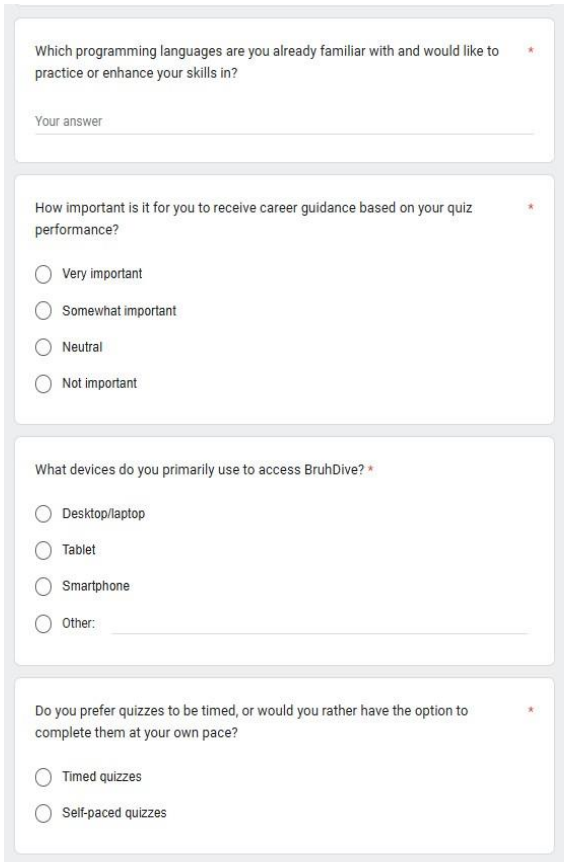
☐ Record **tokkiehehehe@gmail.com** as the email to be included with my response

Name *

Your answer

What is your current experience level in coding? *

- ◯ Beginner
- ◯ Intermediate
- ◯ Advanced
- ◯ Expert



Which programming languages are you already familiar with and would like to practice or enhance your skills in? *

Your answer

How important is it for you to receive career guidance based on your quiz performance? *

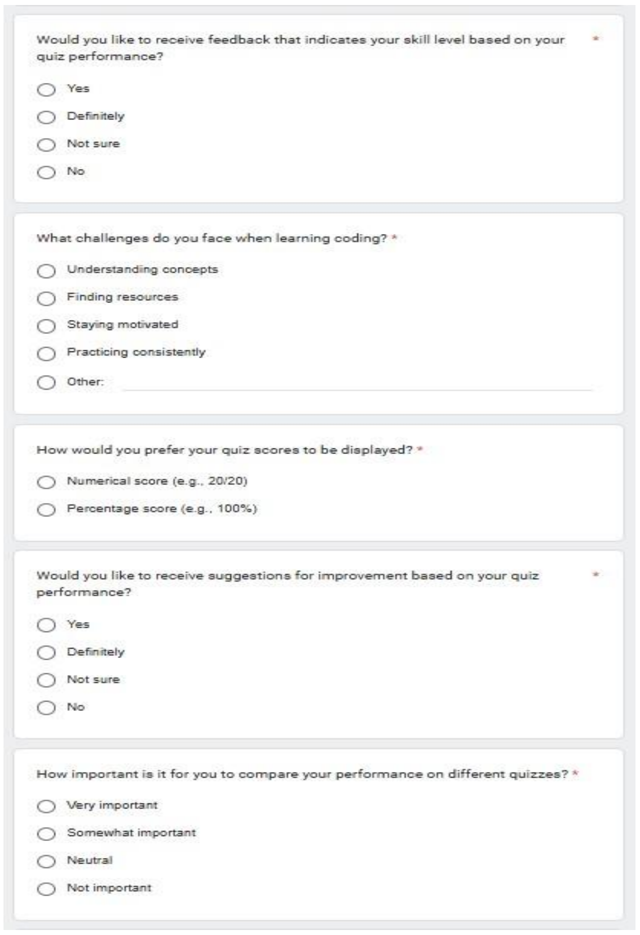
- ◯ Very important
- ◯ Somewhat important
- ◯ Neutral
- ◯ Not important

What devices do you primarily use to access BruhDive? *

- ◯ Desktop/laptop
- ◯ Tablet
- ◯ Smartphone
- ◯ Other:

Do you prefer quizzes to be timed, or would you rather have the option to complete them at your own pace? *

- ◯ Timed quizzes
- ◯ Self-paced quizzes



Would you like to receive feedback that indicates your skill level based on your quiz performance? *

- ◯ Yes
- ◯ Definitely
- ◯ Not sure
- ◯ No

What challenges do you face when learning coding? *

- ◯ Understanding concepts
- ◯ Finding resources
- ◯ Staying motivated
- ◯ Practicing consistently
- ◯ Other:

How would you prefer your quiz scores to be displayed? *

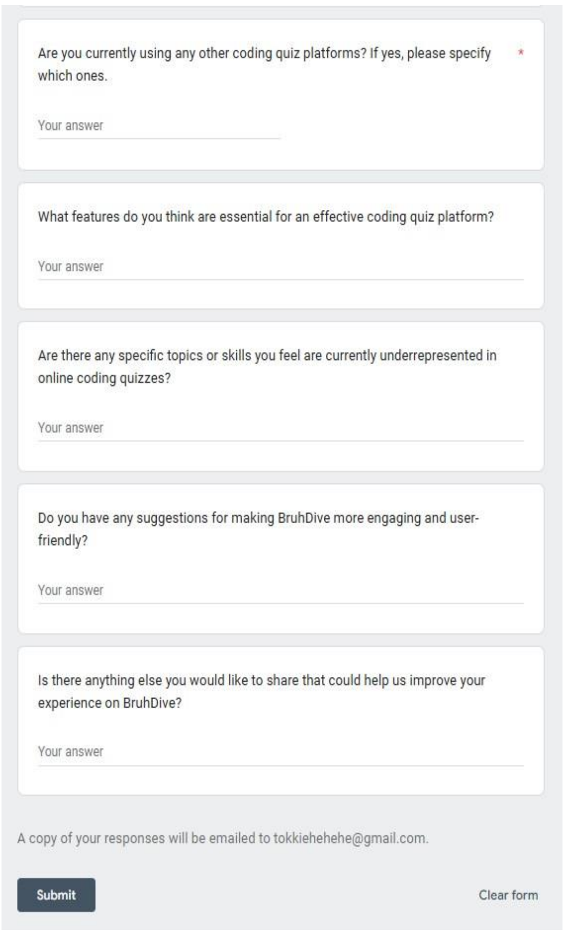
- ◯ Numerical score (e.g., 20/20)
- ◯ Percentage score (e.g., 100%)

Would you like to receive suggestions for improvement based on your quiz performance? *

- ◯ Yes
- ◯ Definitely
- ◯ Not sure
- ◯ No

How important is it for you to compare your performance on different quizzes? *

- ◯ Very important
- ◯ Somewhat important
- ◯ Neutral
- ◯ Not important



Are you currently using any other coding quiz platforms? If yes, please specify which ones. *

Your answer

What features do you think are essential for an effective coding quiz platform?

Your answer

Are there any specific topics or skills you feel are currently underrepresented in online coding quizzes?

Your answer

Do you have any suggestions for making BruhDive more engaging and user-friendly?

Your answer

Is there anything else you would like to share that could help us improve your experience on BruhDive?

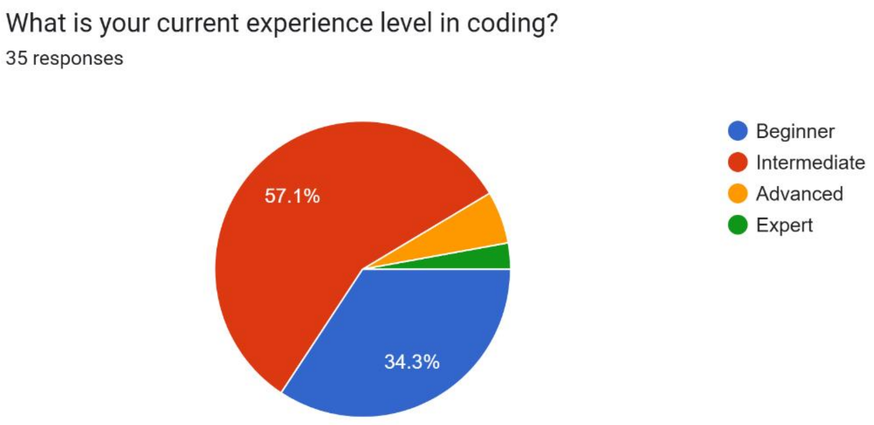
Your answer

A copy of your responses will be emailed to tokkiehehehe@gmail.com.

Submit Clear form

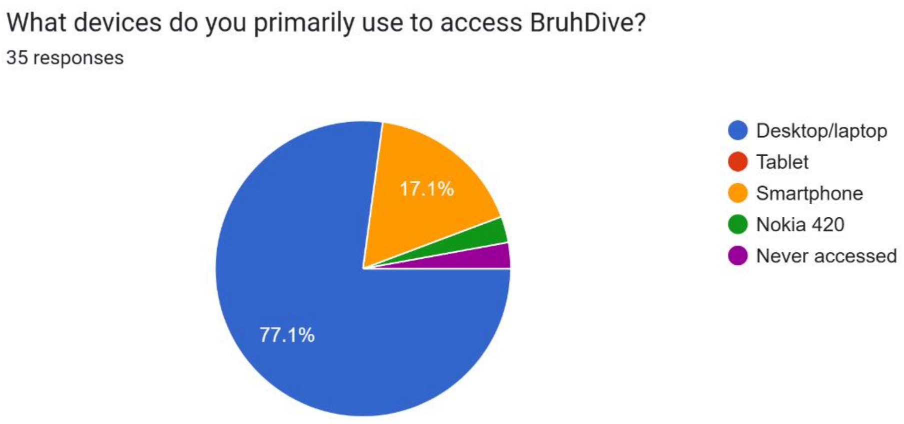
1. Experience Level:

   Participants could identify as Beginner, Intermediate, Advanced, or Expert. This helped us tailor quiz difficulty levels accordingly.



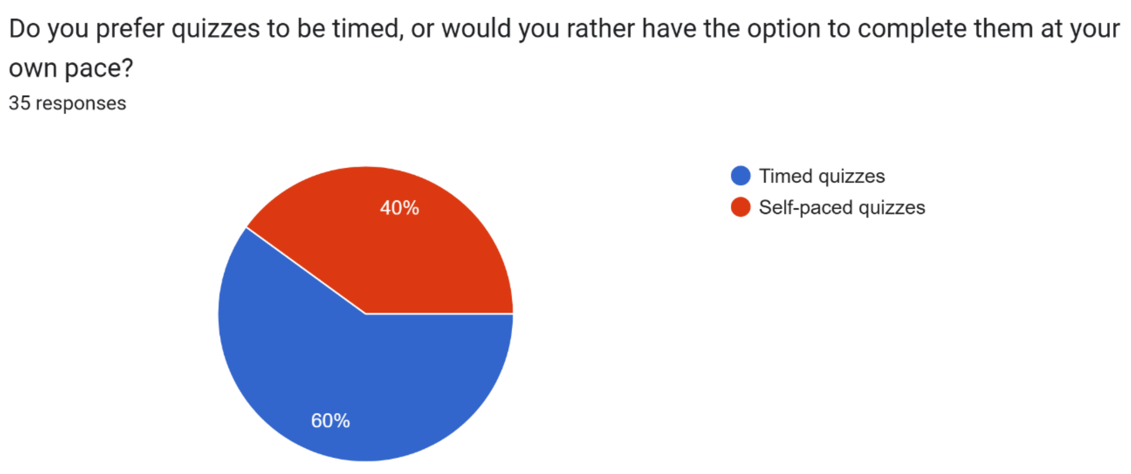
2. Device Accessibility:

   Respondents selected the devices they primarily use for coding practice—desktop, tablet, or smartphone. This informed the responsive design strategy of BruhD



3. Quiz Preferences:

   Options for timed vs. self-paced quizzes were presented to understand learning comfort.



Data Analysis

- User Experience Levels:
  The responses spanned all levels, with the highest concentration among Beginner and Intermediate users.
- Language Interest:
  High demand was observed for languages such as Python, JavaScript, Java, and more guiding the initial language modules on BruhDive.
- Device Preferences:
  Most users preferred desktop environments for learning, followed by smartphones, influencing our mobile responsiveness and layout design.
- Preferred Quiz Format:
  Users leaned slightly toward self-paced quizzes, indicating a preference for a relaxed learning environment. However, a portion also showed interest in timed assessments, suggesting the inclusion of both formats.

Key Takeaways

- Feature Prioritization: Career guidance and personalized feedback should be integral to BruhDive.
- Language Selection: Prioritize beginner-friendly languages like Python and JavaScript.
- Device Optimization: Focus on desktop-first design while maintaining mobile accessibility.
- Learning Challenges: Include encouragement, clarity, and conceptual help in the platform's UI/UX.

By merging these survey results with existing research trends, BruhDive ensures its approach is both evidence-based and user-aligned. The survey responses validated the need for a platform that combines structured coding evaluation with user-friendly features and domain-based insights.

3. ACQUISITION OF KNOWLEDGE

The development of **BruhDive – An Online Quiz Platform** was an insightful journey that involved deep understanding and practical application of modern web technologies. This project integrates a responsive, single-page frontend application built with React and Vite, along with a lightweight Node.js backend used primarily for third-party API communication. Throughout the development process, various technical and conceptual learnings were acquired, spanning from user interface design to backend integration and dynamic content generation.

3.1. FRONTEND DEVELOPMENT – REACT + VITE

The frontend of BruhDive is developed using React, a declarative, component-based JavaScript library for building user interfaces. React's efficient rendering using the Virtual DOM and its support for modular design enabled the creation of reusable components for quiz layouts, scorecards, analytics, feedback sections, and more.

To support faster development and performance-optimized builds, Vite was used as the frontend build tool. Vite offers modern features like hot module replacement (HMR), ES module support, and lightning-fast startup. Unlike older bundlers such as Webpack, Vite compiles modules on-demand, making it ideal for a development environment that prioritizes speed and efficiency.

Key Concepts Learned:

- React Hooks for state and lifecycle management.
- Component reusability and prop drilling.
- Conditional rendering and routing using react-router-dom.
- Handling local storage for storing user quiz history (since there's no DB).
- Integration with external APIs using axios.

3.2. BACKEND DEVELOPMENT – NODE.JS (API)

The backend for BruhDive is minimal yet functional. It is developed using Node.js, a powerful server-side runtime that handles asynchronous tasks efficiently. In this project, Node.js is used specifically to handle API calls for the "Test Yourself" section. This section relies on AI-generated questions using third-party services like DeepInfra or OpenAI.

No database is used in BruhDive. Instead, user quiz data such as selected answers, scores, and feedback are temporarily stored on the client-side using browser local storage. This makes the platform a true "grab-and-go" system with no sign-in or data persistence, emphasizing simplicity and accessibility.

Key Concepts Learned:

- Express server setup for routing and API endpoint creation.
- Integrating with external AI APIs using secure API keys.
- Handling API responses, fallback conditions, and error management.
- Using dotenv to secure and manage environment variables.

3.3. ARCHITECTURE DESIGN & UI EXPERIENCE

A major focus of BruhDive was on **user experience (UX)** and **intuitive UI design**. The platform is built as a single-page application (SPA), ensuring smooth transitions between pages like Home, About, Quiz, Support, Resources, and Results.

Each quiz section is interactive and responsive, with instant result display and visual feedback such as:

- Score breakdown out of 25 questions.
- Performance level (e.g., Beginner, Intermediate, Prodigy).
- Topic-based recommendations based on performance.
- A results history panel powered by local storage.

Tools and Technologies:

- CSS for layout and custom styling.
- react-icons and UI libraries for enhancing visual elements.
- Fallback logic for "Test Yourself" quiz using hardcoded questions if API fails.

3.4. RESOURCE HUB INTEGRATION

BruhDive also includes a Resources section that connects users to curated external links including:

- Roadmaps for developers.
- Free and paid courses.
- Hackathons and event portals.
- Certification programs.
- AI tools, project ideas, and DSA materials.

These links are designed to support a learner's journey by giving direct access to trusted learning resources—eliminating the need for extensive online searches.

3.5. SUPPORT & FEEDBACK MECHANISM

BruhDive features a Support page which includes both a Contact Form and a Feedback Form. These forms are connected using EmailJS, allowing messages and feedback to be sent directly to both the user's email and the project admin email.

Key Concepts Learned:

- Implementing email sending via EmailJS API.
- Managing multiple tab-based forms (Contact & Feedback).
- Sending structured feedback, including user ratings and messages.

3.6. CONCLUSION

The creation of BruhDive provided valuable exposure to full-stack development without a database dependency. From dynamic quiz rendering to performance evaluation and AI-based testing, each feature was designed to be lightweight, user-friendly, and educational.

In the absence of a backend database or user authentication system, browser storage mechanisms and intelligent API usage played a crucial role in building a fully functional, interactive quiz experience.

4. DOMAIN KNOWLEDGE

4.1. INDUSTRY DOMAIN – EDTECH / E-LEARNING PLATFORMS

BruhDive operates in the fast-growing EdTech (Educational Technology) domain, specifically within the niche of online technical assessments and skill guidance. With the global rise in remote learning and self-driven career building, platforms like BruhDive are critical in helping learners evaluate their skills, track progress, and receive guided recommendations.

The platform is built for aspiring developers, students, and tech enthusiasts who want structured, performance-based evaluations across programming languages like Java, JavaScript, SQL, Python, and more. Instead of offering full tutorials or deep theory, BruhDive focuses on gamified quizzes and clear performance insights to enable users to learn by testing.

One of the standout features is performance-level classification. After every quiz, the user is shown what level they currently fall into—such as:

- Noob Mode
- Newbie
- Beginner
- Enthusiast
- Intermediate
- Advanced
- Code Master

This gamified progression system motivates users to improve their performance and move up the ladder. It's not just about getting a score—it’s about understanding where you stand, and what you can do to improve.

Unlike traditional EdTech systems that rely heavily on user accounts or databases, BruhDive is fully browser-based, with all progress stored locally. The backend is built with Node.js APIs, and the frontend uses React + Vite, ensuring fast load times and a responsive, modern user interface.

BruhDive blends quiz-based assessment, real-time performance tracking, and career path suggestions—all in a lightweight, no-login, instant-access environment.

4.2. ADVANTAGES OF DOMAIN KNOWLEDGE

1. **Understanding Learner Behavior and Motivation**

   In the EdTech space, motivation is key. By knowing that learners enjoy tracking growth and achieving ranks, BruhDive's level system becomes a powerful tool. Users aren't just quizzed—they are graded into levels, which pushes them to try again, score higher, and feel accomplished.

2. **Faster and Focused Development**

   With clear awareness of how modern learners interact with EdTech platforms, features like:

   • Local storage-based quiz history

   • Domain-wise performance breakdown

   • Career suggestions based on quiz scores

   …were integrated smoothly, without needing user accounts or database connections.

3. **Targeted Feature Set with Lower Costs**

   Domain knowledge helps avoid unnecessary complexity. For example:

   • No user signup/login → No database maintenance.

   • Results stored in browser → Instant feedback and less backend load.

   • AI-powered quiz generation (with fallback) → Keeps things dynamic without overengineering.

4. **Awareness of Competitors and Market Gaps**

   Most platforms either teach or test, but not both in a balanced way. BruhDive fills this gap by offering:

   • Structured 25-question quiz sets across multiple languages.

   • Performance levels that adapt to quiz score.

   • An AI-driven "Test Yourself" mode to discover one's best-fit tech domain.

5. **Support for Career Discovery and Learning Pathways**

   Understanding that tech learners often don't know which field suits them, BruhDive provides career direction based on quiz performance across domains like:

   • Cybersecurity

   • Web Development

   • DevOps

   • Data Structures, etc.

6. This helps bridge the gap between skill testing and career exploration, without claiming to offer full career guidance.

5. SYSTEM STUDY

5.1. BENEFITS OF PROPOSED SYSTEM

1. **Interactive Quiz-Driven Learning**
    - BruhDive leverages quiz-driven learning, where users actively engage through solving quizzes instead of passively consuming content. This method enhances retention and makes learning more hands-on, especially for those preparing for tech careers.
2. **Performance-Based Feedback**
    - Rather than overwhelming users with too much analysis, BruhDive gives a clear strength assessment by assigning a level (e.g., Beginner, Enthusiast, Code Master) after each quiz. This makes it simple for users to gauge where they stand.
3. **Topic Recommendations Based on Level**
    - Instead of highlighting weaknesses directly, BruhDive provides smart topic suggestions aligned with the user's current level. These recommended topics help users move forward confidently without discouragement.
4. **No Login or Setup Needed**
    - BruhDive is fully browser-based and doesn't require account creation or setup. This allows users to jump straight into learning and evaluation—ideal for quick skill checks and spontaneous practice sessions.
5. **Language-Based Quiz Organization**
    - Quizzes are organized by programming language (e.g., Python, JavaScript), not by difficulty. This makes the platform simple and beginner-friendly. Users select the language they want to test, and the system takes care of the rest.
6. **Accessible and Lightweight**
    - Built using **React + Vite** on the frontend and **Node.js** for backend APIs, BruhDive is optimized for performance and speed. The lack of a heavy database or login system keeps the experience smooth and accessible from any device.
7. **AI-Powered Career Guidance Mode**
    - BruhDive also includes a special "Test Yourself" mode powered by AI. This evaluates a user's aptitude across tech domains like Web Development, Cybersecurity, DevOps, etc., and suggests a fitting career direction based on domain-wise performance.

6. PROBLEM DEFINITION & SCOPE OF PROJECT

6.1. PROBLEM DEFINITION

- This platform is developed as a standalone web-based quiz application and does not extend or modify any pre-existing application.
- There is no issue of maintaining legacy systems or compatibility concerns. The key focus is to provide engaging, insightful, and level-based quiz experiences directly through a modern browser, without requiring app downloads or user logins.
- The goal is not just to conduct quizzes but to help users identify their current skill level, explore multiple programming domains, and get guided topic recommendations based on their performance.

6.2. OBJECTIVE

- The main objective of the BruhDive project is to deliver a simple yet powerful online platform for users to test their programming knowledge across different languages and domains.

- The platform offers 25-question structured quizzes per language, with results that categorize users into performance levels like “Noob Mode” to “Code Master.”

- Users can also explore the “Test Yourself” AI-powered mode, where they are assessed across various domains (e.g., Cybersecurity, Web Dev, DSA), and are then recommended a potential tech path based on their strengths.

- The platform also suggests learning topics based on performance level to support continued learning.

6.3. PROPOSED SYSTEM

- The proposed system is a **fully browser-based web application**, accessible on any modern desktop or mobile browser. It does not require installation from app stores.

- Developed using **React + Vite** on the frontend and **Node.js** for backend APIs, the system leverages **local storage** to manage quiz history and results instead of any external database or user login.

- The application is designed to be lightweight, fast, and educational—with no dependency on cloud databases or account management.

- Upon launching the app, users can:

    - Select a programming language quiz (e.g., Python, JavaScript, Java).
    - Attempt a timed 25-question quiz.
    - Instantly receive their score, assigned level, strengths, and topic recommendations.
    - Use the AI-powered career discovery mode for domain-specific evaluation.

- The system makes use of **fallback questions** when API-generated questions are unavailable, ensuring uninterrupted experience.

- Since results are stored in-browser, users can revisit and track their quiz performances without requiring an account.

7. REQUIREMENT ANALYSIS

7.1. ANALYSIS

- Requirements are the detailed specifications of the functionalities a software system must offer and the constraints within which it should operate.
- These may vary from high-level expectations to specific technical or functional needs.
- Requirement Engineering is the process of identifying and defining what the user expects from the system and the limitations under which it should be developed and maintained.

Requirements for the BruhDive platform can be categorized into the following:

1. **User Requirements**
   - The system should allow users to take quizzes in different programming languages directly from their browser.
   - No signup/login should be required to use the platform.
   - Users should be able to view their quiz results instantly, including performance levels and recommended topics.

2. **Functional Requirements**
   - The platform must support the following core functionalities:
     - Display quiz options (by programming language).
     - Conduct 25-question quizzes with scoring.
     - Show result level (e.g., Beginner, Advanced, Code Master).
     - Provide topic recommendations based on score.
     - Enable AI-powered "Test Yourself" quiz mode across domains.
     - Store quiz history using local browser storage.

3. **Non-Functional Requirements**
   - The system should be lightweight and fast-loading on both desktop and mobile browsers.
   - It should remain functional even without database or user authentication.
   - It must handle errors gracefully (e.g., fallback if the AI API fails).
   - User data like quiz scores must be stored locally in the browser for privacy and offline access.

4. **System Requirements**
    - BruhDive must be accessible through modern web browsers on desktops and smartphones.
    - No installation or download is needed—purely web-based.
    - Internet access is required to load the web application and fetch AI-generated content when available.
    - The frontend should run using React + Vite and connect to a backend powered by Node.js/Express.js.

7.2. FEASIBILITY STUDY

Each software project must undergo a feasibility study to determine its viability from various angles. For BruhDive, we consider **technical** and **economic** feasibility.

1. **Technical Feasibility**
    - BruhDive is built using industry-standard technologies that are widely supported and easy to maintain:
        - **Frontend**: React.js + Vite
        - **Backend**: Node.js + Express
        - No mobile OS dependency — works on all major browsers (Chrome, Firefox, Edge, Safari).
    - Local storage is used instead of databases, simplifying deployment and reducing maintenance.
    - The platform works even without user accounts, making it lightweight and user-friendly.
2. **Economic Feasibility**
    - Since BruhDive uses open-source tools and doesn’t rely on paid hosting or database solutions, the project is highly cost-effective.
    - API usage (e.g., for AI-generated questions) can be scaled or fallback questions used to avoid downtime and costs.
    - Using browser local storage reduces backend hosting needs and operational expenses.

7.3. HARDWARE & SOFTWARE REQUIREMENTS

Minimum hardware requirements

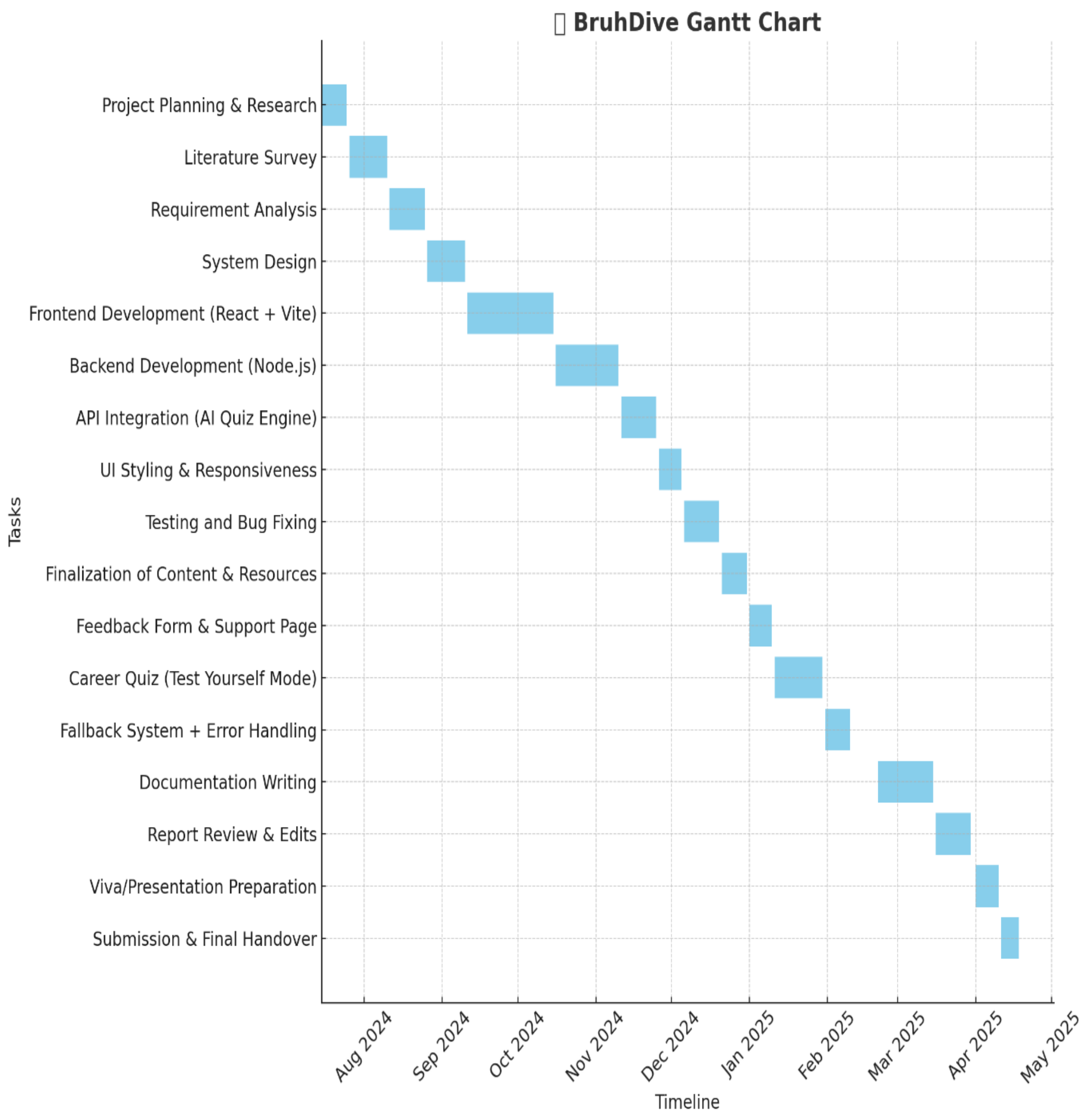
- **Processor**: Intel i3 or equivalent
- **RAM**: Minimum 2GB recommended for smooth browser performance
- **Hard Disk**: No installation required, minimal browser cache used

Software Requirements

- **Operating System**: Any OS that supports modern web browsers (Windows, macOS, Linux, Android, iOS)
- **Browser**: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari (latest versions recommended)
- **Internet Connection**: Required for loading the platform and generating quizzes

8. ESTIMATION AND PLANNING

8.1. GANTT CHART,



9. METHODOLOGY

9.1. CHOSEN SDLC MODEL: SPIRAL MODEL

The Spiral Model is the most suitable development methodology for the BruhDive project — a modern, browser-based online quiz platform built using React and Node.js. While not AI-powered, BruhDive emphasizes dynamic user interaction, performance-based feedback, and structured evaluation across programming domains. The Spiral Model's iterative nature aligns perfectly with BruhDive's phased, evolving development process.

9.2. OBJECTIVES OF SPIRAL MODEL

- Enable iterative and modular development of quiz features, resource pages, and performance logic.
- Allow flexibility for changes based on user testing and feedback.
- Incorporate early risk analysis to tackle issues with fallback logic, UI responsiveness, or third-party dependencies.
- Improve design through user input, especially during testing and revision phases.
- Ensure that major modules (like quiz logic, result evaluation, and recommendation engine) are stable and tested before moving forward.

9.3. WHY SPIRAL MODEL SUITS BRUHDIVE

1. Iterative Feature Rollouts

BruhDive's development involved multiple modules like:

- Structured language-based quizzes
- "Test Yourself" mode for domain exploration
- Result analysis with performance levels
- Local storage for quiz history
- Resource and feedback sections

Each of these components was designed, tested, and refined over time. The **spiral model allows for incremental improvements** during every cycle, making it ideal for such modular development.

2. Evolving Requirements

During development, new ideas were introduced:

- Gamified feedback based on score

- Fallback question support in case of errors
- Roadmap/resource recommendations

The **flexibility** of the Spiral Model allowed the BruhDive team to accommodate evolving requirements without starting over, unlike rigid models like Waterfall.

3. Built-In Risk Management

BruhDive had some uncertainties in:

- Quiz question reliability from external APIs
- Browser-only result tracking (no user accounts or database)
- Responsive UI behavior across devices

The Spiral Model emphasizes **early risk assessment and mitigation**, making it perfect for handling technical or design uncertainties — especially when working without traditional backend features like login systems or databases.

4. Continuous Feedback Integration

User feedback from early testing helped refine:

- UI layout
- Quiz result logic
- Resource page structuring

The Spiral Model includes a dedicated **customer evaluation phase** in every loop, ensuring BruhDive could respond to feedback in real-time without waiting for full system completion.

5. Cost & Time Efficiency

By allowing partial system development and early validation of core modules, the Spiral Model:

- **Reduced rework**
- Improved focus on priority features
- Balanced time and effort effectively

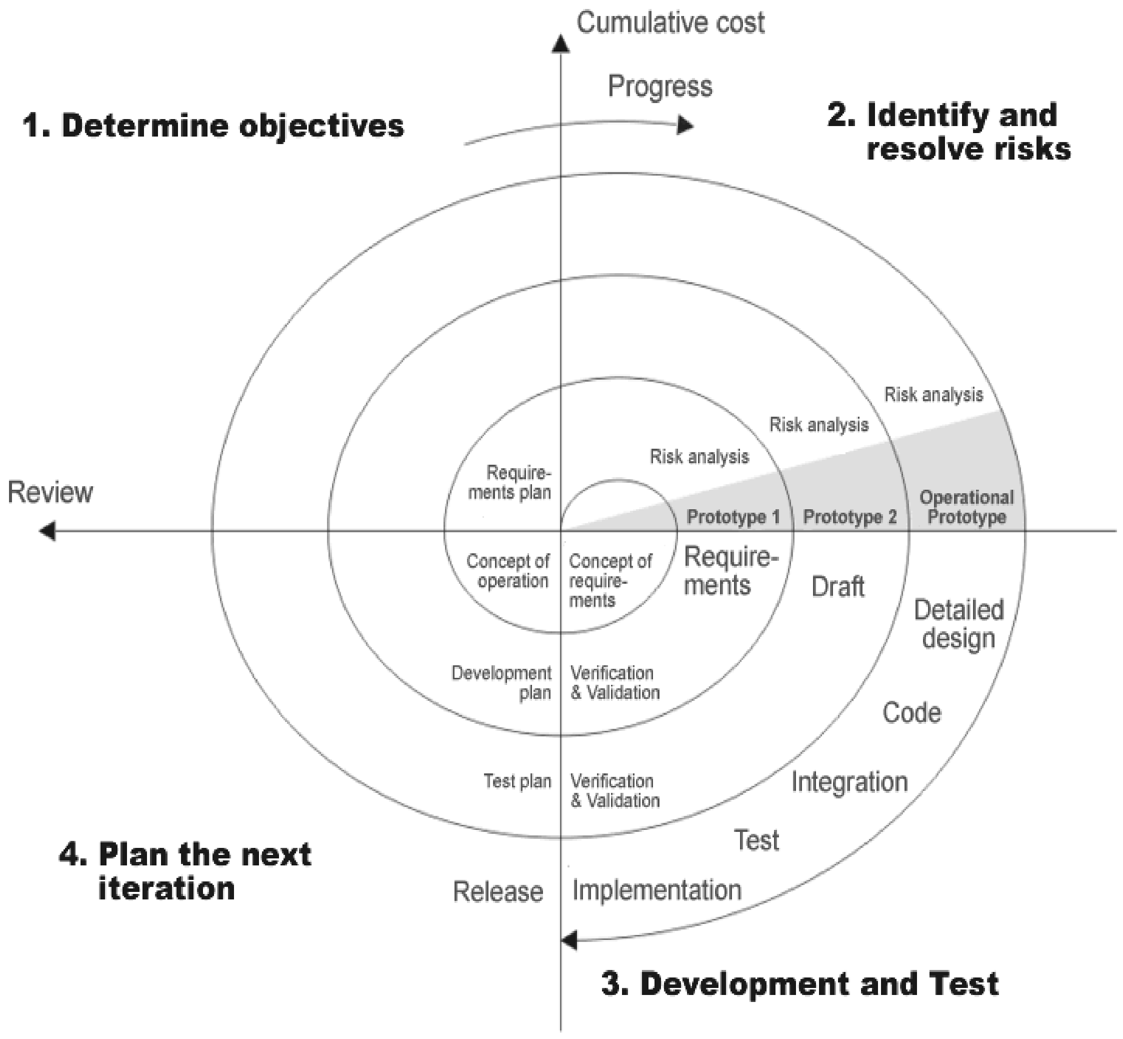
This was especially useful since BruhDive was developed without hosting costs or user authentication overhead.

9.4. ADVANTAGES OF USING SPIRAL MODEL FOR BRUHDIVE

- Risk Handling: Proactively addressed uncertainties like no-database architecture, offline fallback, and local storage behavior.
- Iterative Improvement: Allowed for the step-by-step refinement of features like performance levels, quiz categories, and results UI.
- Early Testing: Each module (frontend, quiz logic, results page) was tested independently before full integration.
- Adaptability: New ideas like domain-wise quiz sets, dynamic feedback, and recommendation buttons were added during development without disrupting the process.
- Scalable for Future Enhancements: Spiral Model allows future additions (like user login, hosting, or analytics) to be integrated smoothly in new cycles.

9.5. PHASES OF SPIRAL MODEL

- Planning Phase
  Project goals, tech stack (React + Node.js), features like quizzes, result analysis, and career suggestions were planned.
- Risk Analysis Phase
  Identified risks like API failure, no backend database, and user interface issues. Solutions like fallback questions and localStorage were planned.
- Engineering Phase
  Frontend and backend were developed. Key features like quiz system, result evaluation, and resource linking were implemented.
- Customer Evaluation Phase
  Testing and feedback were done. Bugs were fixed, UI improved, and final adjustments made based on user reviews.


Cumulative cost
Progress
1. Determine objectives
2. Identify and resolve risks
Risk analysis
Risk analysis
Risk analysis
Require-ments plan
Review
Prototype 1
Prototype 2
Operational Prototype
Concept of operation
Concept of require-ments
Require-ments
Draft
Detailed design
Development plan
Verification & Validation
Code
Test plan
Verification & Validation
Integration
Test
4. Plan the next iteration
Release
Implementation
3. Development and Test

10. OPERATING TOOLS AND DEVELOPMENT ENVIRONMENT

10.1. REACT WITH VITE (FRONTEND DEVELOPMENT)

For building the frontend of BruhDive, React is used in combination with Vite as the development environment. React is a powerful JavaScript library for creating dynamic user interfaces, and Vite is a next-generation frontend build tool that offers lightning-fast development with native ES modules and hot module replacement (HMR).

The choice of React enables component-based architecture, allowing the platform to efficiently manage different sections like:

- Homepage
- About Page
- Quiz Page
- Resources Page
- Support Page (Contact & Feedback)
- View Result Page

Vite enhances the development experience with:

- Fast startup and rebuild times
- Instant hot updates
- Optimized production builds

The UI is built with plain CSS, focusing on simplicity and performance, making the platform accessible and lightweight.

10.2. NODE.JS (BACKEND API)

The backend of the BruhDive project is built using Node.js. It handles only one core responsibility: interfacing with the AI API to generate domain-wise questions and career guidance in the "Test Yourself" quiz feature.

Key aspects of the Node.js backend:

- Fetches dynamic questions from the AI model via API key
- Handles fallback mechanisms in case the API fails
- Returns career guidance based on performance

No database integration is used in this project. Instead, local storage is leveraged on the client-side to retain quiz history for the "View Results" page, ensuring a lightweight and secure experience.

10.3. LOCAL STORAGE (CLIENT-SIDE DATA HANDLING)

To maintain session data and quiz history, the platform uses browser local storage. This allows users to:

- View past results without signing in
- Keep track of quiz attempts during their browser session

Data is retained temporarily unless the browser storage is cleared, supporting the platform's goal of a frictionless "grab-and-go" experience without requiring login or registration.

Supporting Tools

- Visual Studio Code (VS Code): The primary code editor used for development. It offers IntelliSense, Git integration, extensions for React/Node.js, and debugging tools.
- Postman: Used for testing API endpoints and verifying data responses during development.
- Git & GitHub: For version control, collaboration, and source code management.
- Nodemon: Utilized during backend development for automatic server restart on changes.

Why This Stack?

This development environment was chosen to ensure:

- Speed: Vite + React provides a fast dev and build experience.
- Scalability: Component-based frontend structure allows easy future expansion.
- Simplicity: Node.js backend with a single API endpoint reduces complexity.
- User Experience: Local storage provides a seamless experience without requiring authentication.

The result is a responsive, efficient, and accessible platform that meets the goals of interactive skill evaluation and personalized career guidance.

11. DESIGINING

Overview

The designing phase of a software project defines the overall system flow and architecture that will be implemented. For BruhDive, an online quiz platform, the system's design ensures a smooth user experience, clearly defined navigation, and a modular structure to support features like quizzes, career guidance, and results visualization.

We use UML (Unified Modeling Language) to model and represent the system design. UML is a standardized way to visualize the design of a system using various diagrams. These diagrams help in analyzing the static and dynamic behavior of the application, making it easier for developers and stakeholders to understand system functionality.

For BruhDive, the design process follows these key steps:

- Identify all key interfaces/screens like Home, Quiz, View-Result, Support, and Resources.
- Define user interactions using Use Case Diagrams.
- Model data flow and control with Activity Diagrams.
- Represent system structure with Class Diagrams.
- Show communication flow using Sequence Diagrams.
- Outline the development lifecycle using the Spiral Model for iterative design.

11.1. SPIRAL MODEL (SOFTWARE PROCESS MODEL USED)

BruhDive follows the Spiral Model, which is ideal for iterative development and progressive refinement of the system. The Spiral Model supports flexible planning, frequent user feedback, and risk handling—making it perfect for a quiz-based educational product with evolving features.

Phase	Description for BruhDive
Planning	Requirement gathering, quiz formats, UI flow discussions, domain categorization.
Risk Analysis	Evaluate potential failures like API fallback, UI issues.
Engineering	Frontend with React + Vite, backend with Node.js, fallback logic, local storage handling.
Evaluation	Testing quiz functionality, result computation

11.2. USE CASE DIAGRAM

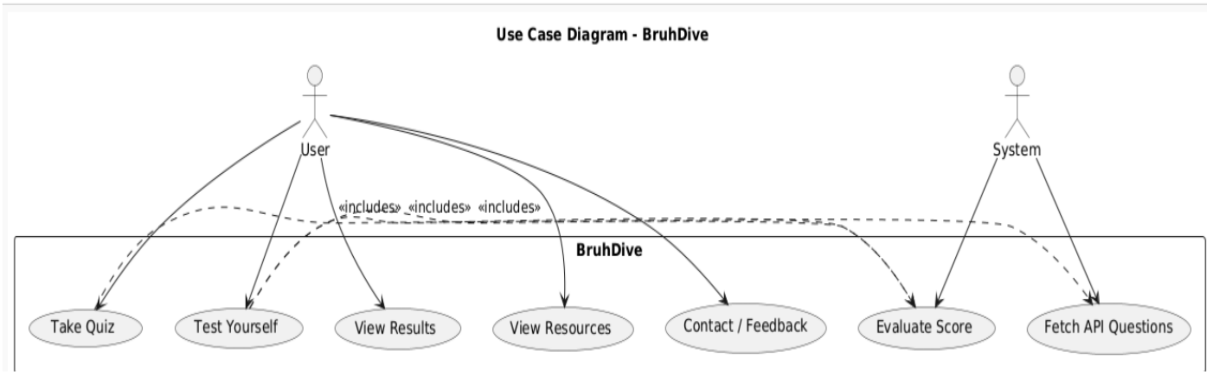
A Use Case Diagram represents how various users (actors) interact with the BruhDive system. The two main actors are:

- User – who takes quizzes, views results, and accesses resources.
- System – handles quiz rendering, score evaluation, and API calls.

Tool Name	Notation	Description
System	Rectangle	Defines the system boundary (BruhDive)
Actor	Stick figure	End-user interacting with the system
Use Case	Oval	Represents actions like "Take Quiz", "View Results"
Association	Line	Connects actor to corresponding use case
Includes	Dashed Arrow	Indicates common functionality reused

Use Cases in BruhDive:

- Take Quiz (for selected language)
- View Results (from local storage)
- Test Yourself (API-driven with career guidance)
- View Resources (roadmaps, internships, tools)
- Contact/Feedback (Support page)



11.3. CLASS DIAGRAM

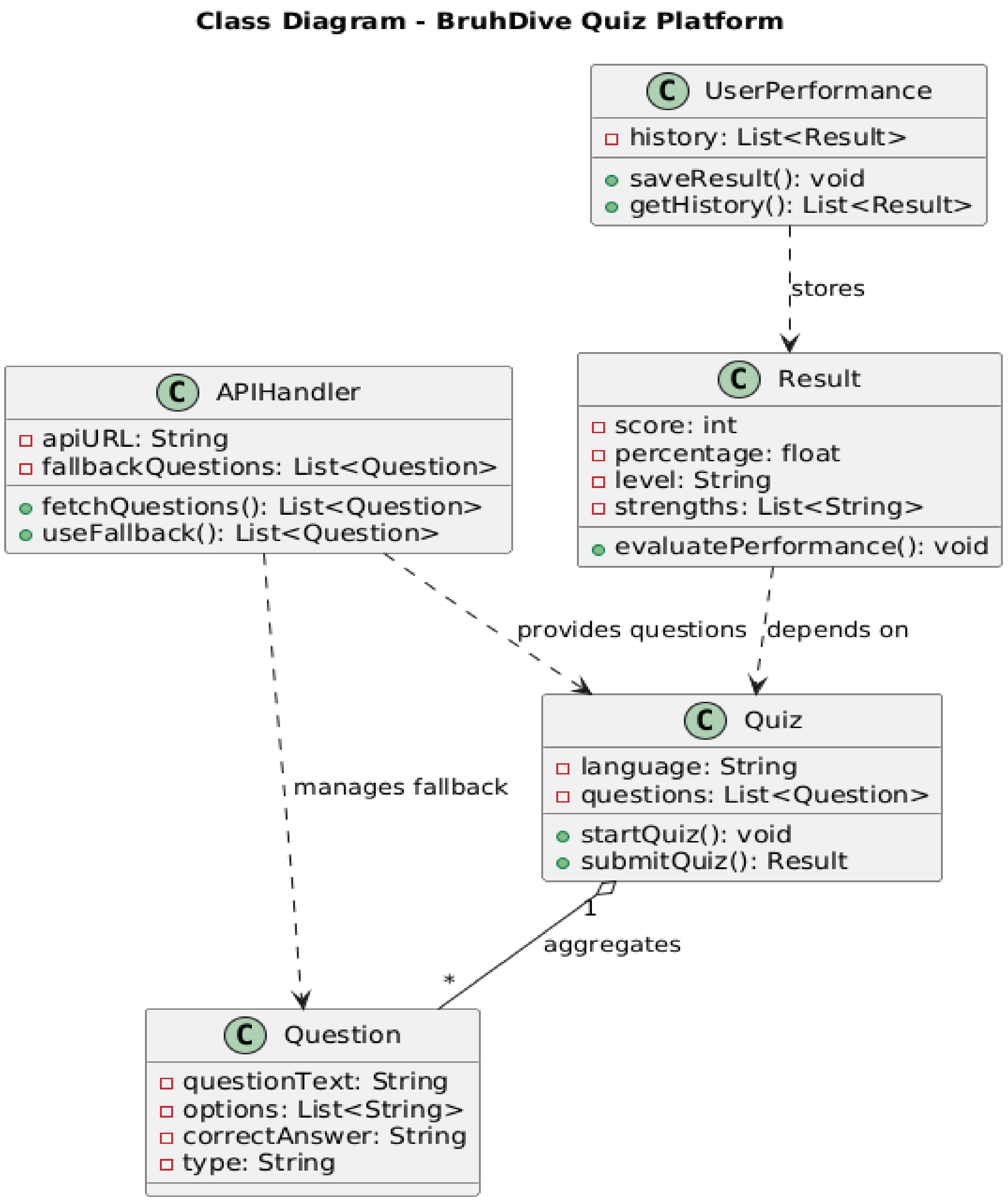
The Class Diagram models the static structure of the BruhDive system—how different components (classes) interact, and what attributes and operations they contain.

Tool Name	Notation	Description
Class	Rectangle	Represents objects like Quiz, Result, UserPerformance
Association	Line	Shows relationships between classes

Aggregation	Hollow diamond	Denotes whole-part relationship (e.g., Quiz has multiple Questions)
Dependency	Dashed arrow	Shows dependency between components (e.g., Result depends on Quiz Evaluation)

Key Classes:

- Quiz (language, questions)
- Question (questionText, options, correctAnswer)
- Result (score, percentage, level, strengths)
- UserPerformance (history saved in localStorage)
- APIHandler (manages fallback and Test Yourself mode)



11.4. ACTIVITY DIAGRAM

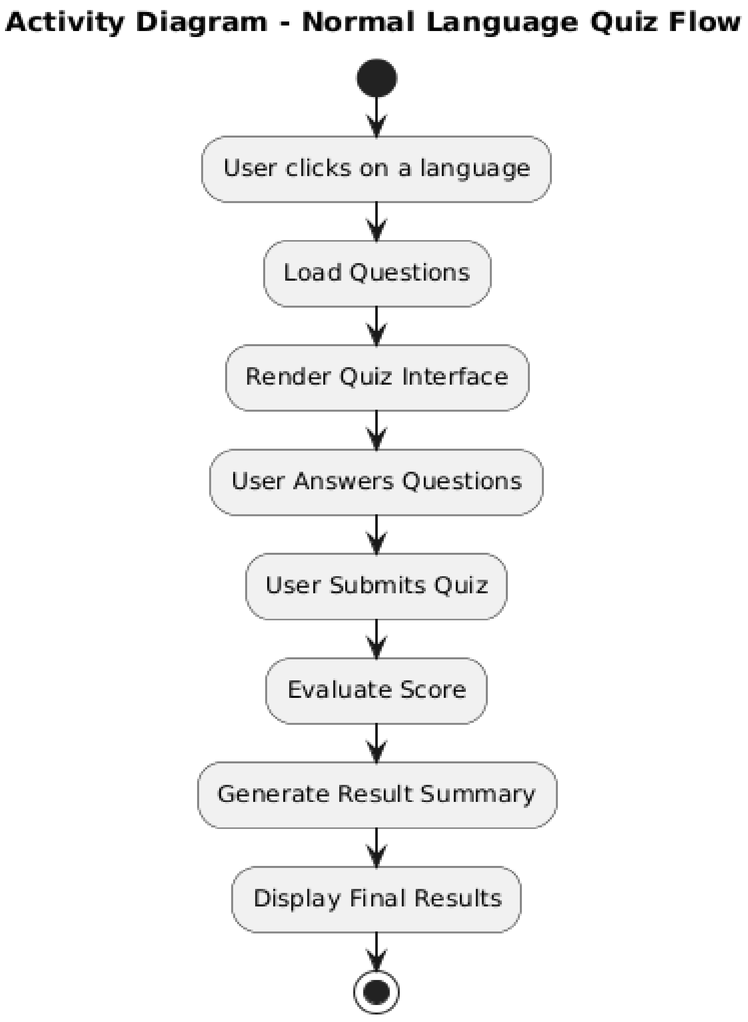
An Activity Diagram visually represents the flow of control or data from one activity to another. It is ideal for modeling user interactions in BruhDive, such as taking a quiz and receiving results.

Tool Name	Notation	Description
Initial State	Solid circle	Entry point of activity
Action State	Rounded rectangle	Represents an action (e.g., Start Quiz, Show Results)
Decision	Diamond	Represents decision points (e.g., Is API Working?)
Final State	Bullseye symbol	Marks end of activity
Flow Arrow	Arrow	Shows transition between activities

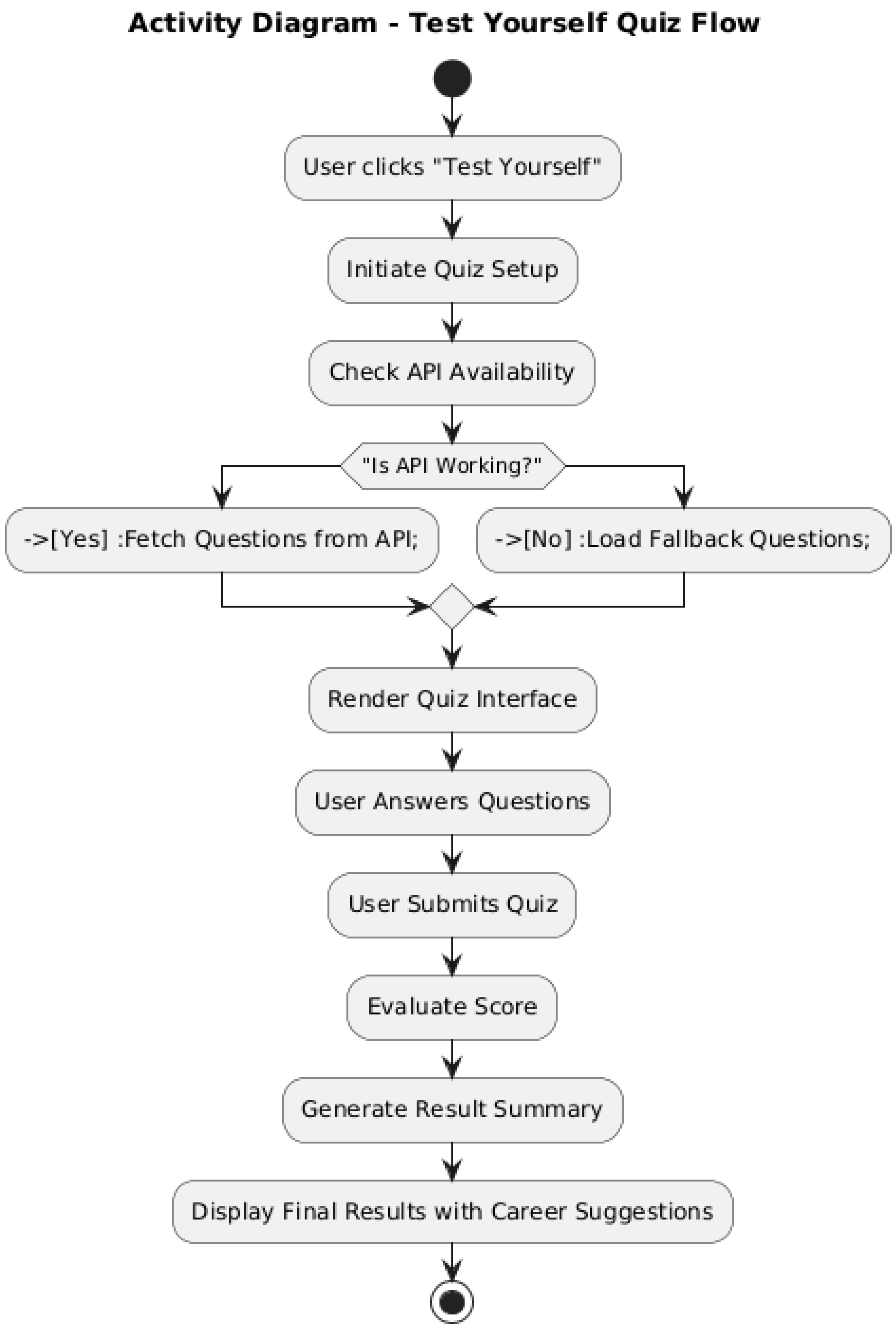
Sample Flow:

- User clicks a language card → Quiz starts → Answering questions → Quiz submitted → Score calculated → Result shown.

A. Activity Diagram - Normal Language Quiz (Hardcoded)



B. Activity Diagram - Test Yourself Quiz (API + Fallback)


Activity Diagram - Test Yourself Quiz Flow
User clicks "Test Yourself"
Initiate Quiz Setup
Check API Availability
"Is API Working?"
->[Yes] :Fetch Questions from API;
->[No] :Load Fallback Questions;
Render Quiz Interface
User Answers Questions
User Submits Quiz
Evaluate Score
Generate Result Summary
Display Final Results with Career Suggestions

11.5. SEQUENCE DIAGRAM

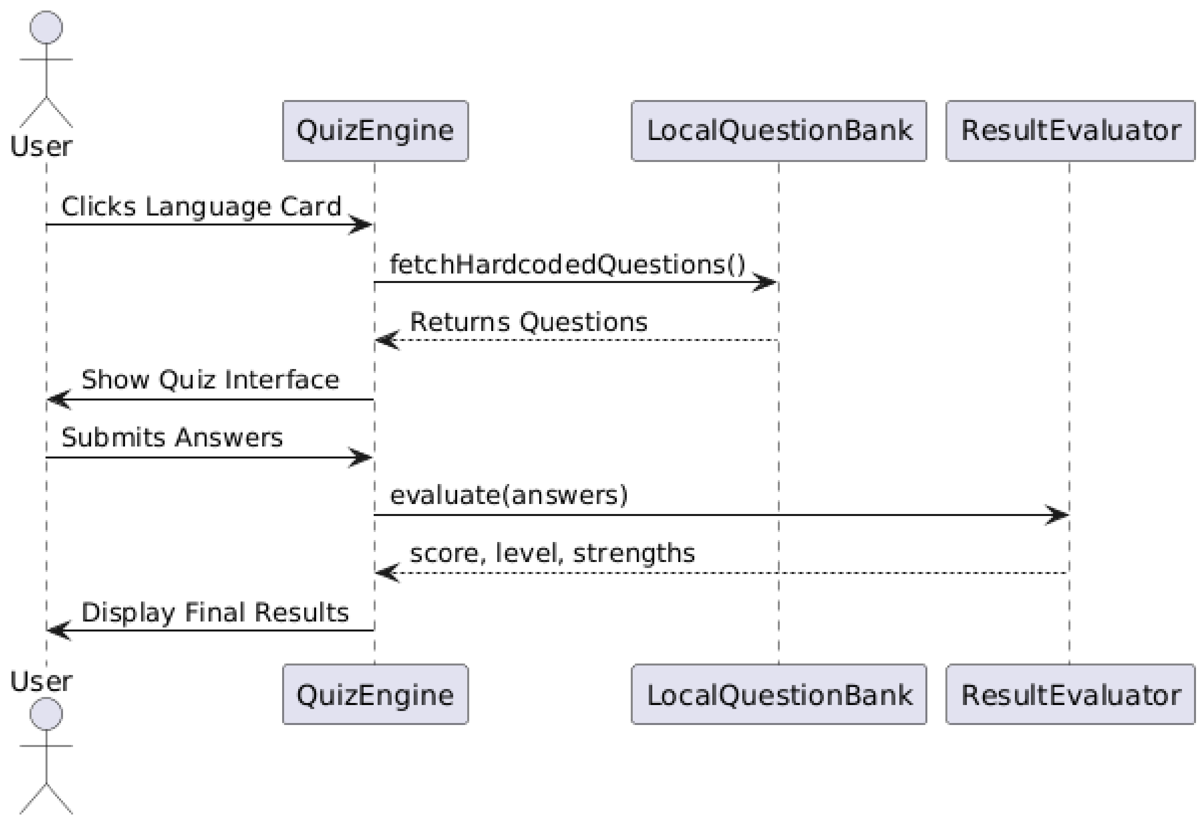
The Sequence Diagram shows the interaction between components over time. It’s useful for visualizing the process of taking a quiz and handling result logic or fallback cases.

Tool Name	**Notation**	**Description**
Object	Rectangle	Represents components (e.g., User, QuizEngine, API, FallbackHandler)
Stimulus	Arrow	Represents messages or function calls between components
Lifeline	Vertical dashed line	Duration an object exists in the process
Activation bar	Thin vertical box	Denotes the period an object is active

Example Interaction (Test Yourself):

- User → clicks "Test Yourself" →
- System → calls API →
- If API fails → fetches fallback →
- Evaluates answers → returns career advice.

Sequence Diagram - Hardcoded Quiz Flow (BruhDive)



Conclusion on Design Choice for BruhDive

Given the structure of BruhDive, a combination of UML-based object-oriented design and the Spiral Development Model is the most suitable:

- Spiral Model handles iterative development and evolving features like the AI-powered test-yourself card.
- UML Diagrams provide clarity in system design, user interaction, and backend logic.
- The design supports scalability (easy to add more quizzes), fallback logic (gracefully handles API failure), and simple user experience (click → quiz → result).

12. IMPLEMENTATION

12.1. SCREENSHOTS

➢ Home Screen

Description:

The Home Screen is the landing page of the BruhDive platform. It includes navigation links to other pages such as About, Support, and Resources.

File: Hero.jsx

```
const messages = [
  "Hey Bro! Wassup!",
  "Welcome to BruhDive, where coding meets chill vibes!",
  "Ready to flex those coding skills? Don't stress—just vibe out and dive in!",
];

const Hero = () => {
  const [currentMessage, setCurrentMessage] = useState(0);
  const [floatingLogos, setFloatingLogos] = useState([]);

  useEffect(() => {
    const interval = setInterval(() => {
      setCurrentMessage((prev) => (prev + 1) % messages.length);
    }, 4000);
    return () => clearInterval(interval);
  }, []);

  return (
    <section className="hero">
      <div className="floating-logos">
        {floatingLogos.map((logo) => (
          <div key={logo.id} style={{ left: `${logo.x}vw`, top: `${logo.y}vh` }}>
            {logo.text}
          </div>
        ))}
      </div>
      <h1>{messages[currentMessage]}</h1>
    </section>
  );
};
```

Description:

- Rotates welcome messages using useEffect.
- Generates floating language names (React, Python, etc.) as ambient visual effects.
- Core part of the **BruhDive's Home Page UI**.

File: Steps.jsx

```
const Steps = () => {
  const steps = [
    { icon: "🔍", title: "1. Choose Your Language" },
    { icon: "📝", title: "2. Take the Quiz" },
    { icon: "📊", title: "3. Get Your Results" },
    { icon: "🚀", title: "4. Improve Your Skills" },
  ];

  return (
    <section className="steps-section">
      {steps.map((step, i) => (
        <div key={i}>
          <span>{step.icon}</span>
          <h3>{step.title}</h3>
        </div>
      ))}
    </section>
  );
};
```

Description:

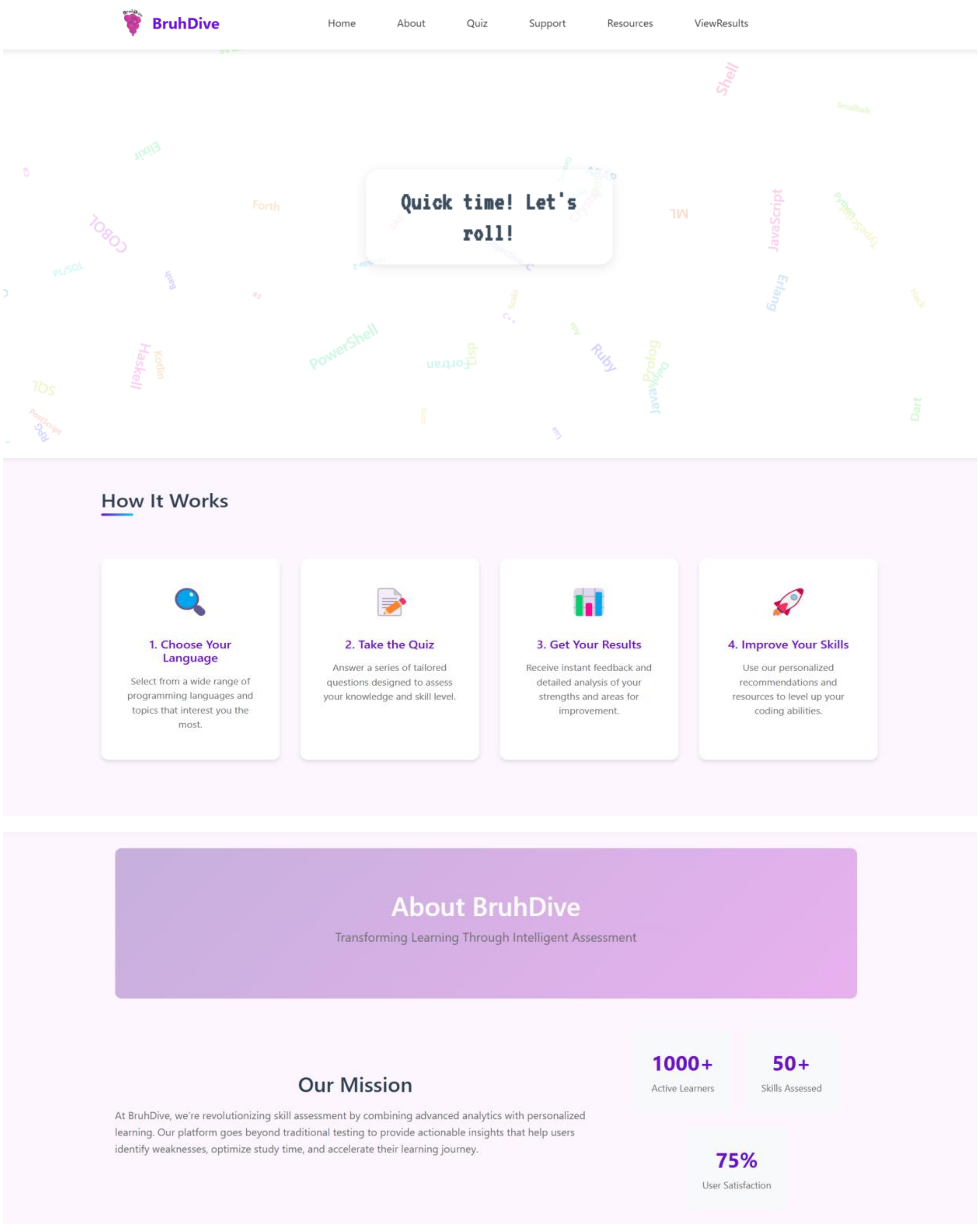
Displays a 4-step process explaining how the platform works — from selecting a quiz to getting improvement tips.

File: Navbar.jsx (Navigation Bar)

```
const Navbar = () => {
  return (
    <nav className="navbar">
      <img src="/BDlogo.png" alt="BruhDive" />
      <button onClick={() => navigate("/")}>Home</button>
      <button onClick={() => navigate("/quiz")}>Quiz</button>
      <button onClick={() => navigate("/support")}>Support</button>
    </nav>
  );
};
```

Description:

Handles navigation across all pages using react-router-dom. Responsive and collapsible for mobile.



BruhDive

Home About Quiz Support Resources ViewResults

Quick time! Let's roll!

How It Works

1. Choose Your Language

Select from a wide range of programming languages and topics that interest you the most.

2. Take the Quiz

Answer a series of tailored questions designed to assess your knowledge and skill level.

3. Get Your Results

Receive instant feedback and detailed analysis of your strengths and areas for improvement.

4. Improve Your Skills

Use our personalized recommendations and resources to level up your coding abilities.

About BruhDive

Transforming Learning Through Intelligent Assessment

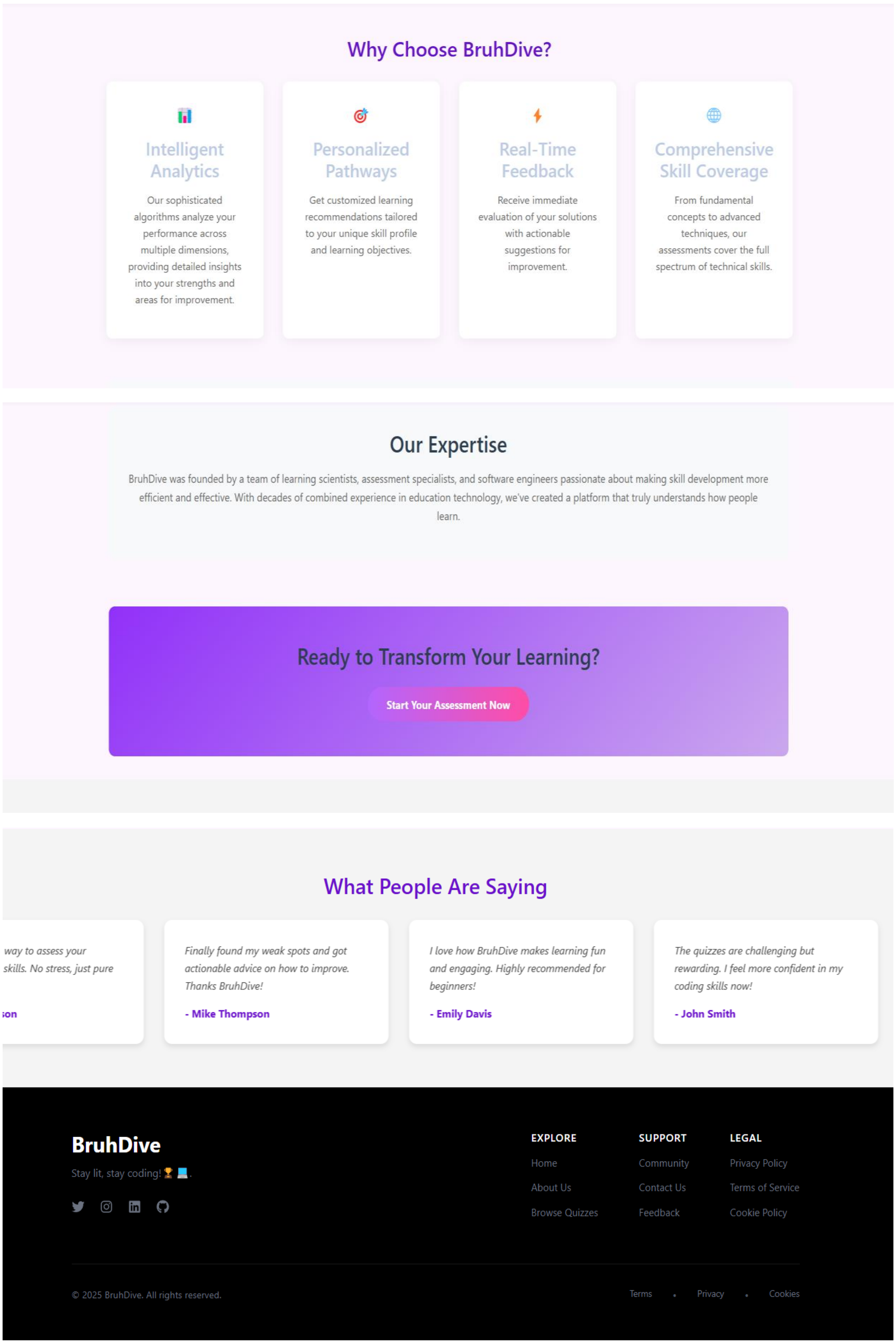
Our Mission

At BruhDive, we're revolutionizing skill assessment by combining advanced analytics with personalized learning. Our platform goes beyond traditional testing to provide actionable insights that help users identify weaknesses, optimize study time, and accelerate their learning journey.

1000+ Active Learners

50+ Skills Assessed

75% User Satisfaction



Why Choose BruhDive?

Intelligent Analytics

Our sophisticated algorithms analyze your performance across multiple dimensions, providing detailed insights into your strengths and areas for improvement.

Personalized Pathways

Get customized learning recommendations tailored to your unique skill profile and learning objectives.

Real-Time Feedback

Receive immediate evaluation of your solutions with actionable suggestions for improvement.

Comprehensive Skill Coverage

From fundamental concepts to advanced techniques, our assessments cover the full spectrum of technical skills.

Our Expertise

BruhDive was founded by a team of learning scientists, assessment specialists, and software engineers passionate about making skill development more efficient and effective. With decades of combined experience in education technology, we've created a platform that truly understands how people learn.

Ready to Transform Your Learning?

Start Your Assessment Now

What People Are Saying

way to assess your skills. No stress, just pure

son

Finally found my weak spots and got actionable advice on how to improve. Thanks BruhDive!

- Mike Thompson

I love how BruhDive makes learning fun and engaging. Highly recommended for beginners!

- Emily Davis

The quizzes are challenging but rewarding. I feel more confident in my coding skills now!

- John Smith

BruhDive

Stay lit, stay coding! 🏆 💻.

EXPLORE	SUPPORT	LEGAL
Home	Community	Privacy Policy
About Us	Contact Us	Terms of Service
Browse Quizzes	Feedback	Cookie Policy

© 2025 BruhDive. All rights reserved.

Terms • Privacy • Cookies

Figure 12.1: Home Page

➢ About

Description:

This page provides background information about the BruhDive platform, its purpose & features.

File: About.jsx

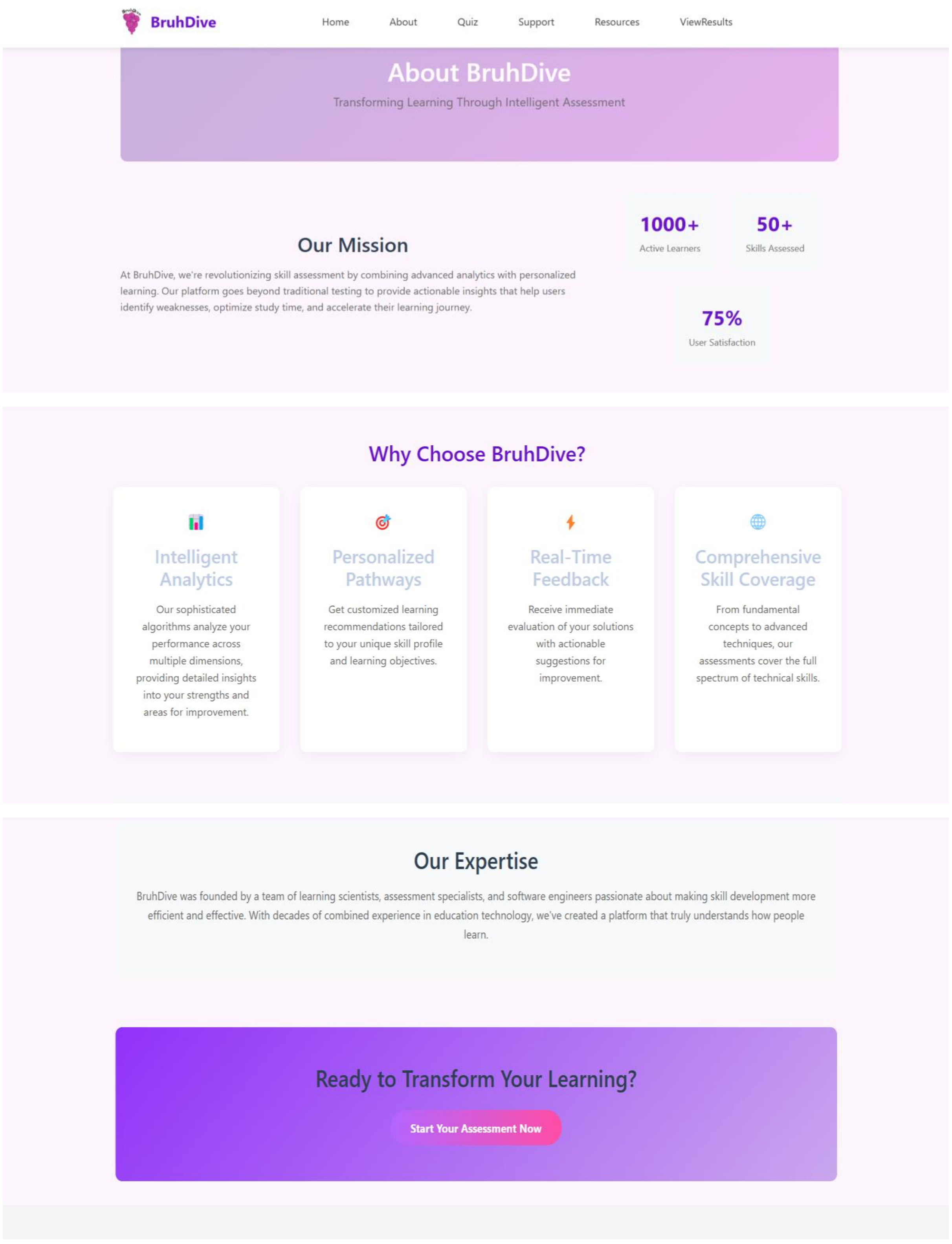
```
import React from "react";
import { useNavigate } from "react-router-dom";

const About = () => {
  const navigate = useNavigate();

  const handlequiz = () => {
    navigate("/Quiz");
    window.scrollTo(0, 0);
  };

  return (
    <div className="about-container">
      <header>
        <h1>About BruhDive</h1>
        <p>Transforming Learning Through Intelligent Assessment</p>
      </header>

      <section>
        <h2>Our Mission</h2>
        <p>
          We're revolutionizing skill assessment by combining analytics with
          personalized learning to help users identify weaknesses and optimize
          study time.
        </p>
      </section>

      <section>
        <h2>Why Choose BruhDive?</h2>
        <ul>
          <li>📊 Intelligent Analytics</li>
          <li>🎯 Personalized Pathways</li>
          <li>⚡ Real-Time Feedback</li>
          <li>🌐 Skill Coverage from Basics to Advanced</li>
        </ul>
      </section>

      <section>
        <h2>Our Expertise</h2>
        <p>
          Founded by learning scientists and developers, BruhDive brings together experience and
innovation to modern skill evaluation.
        </p>
      </section>

      <section>
        <h2>Ready to Transform Your Learning?</h2>
        <button onClick={handlequiz}>Start Your Assessment Now</button>
      </section>
    </div>
  );
};

export default About;
```

Description for Documentation:

The **About Page** introduces BruhDive’s purpose and core features. It outlines:

- Platform mission and goals
- Why learners should use it
- Technical strengths and expertise
- A call-to-action to begin the quiz journey

Navigation is powered by useNavigate from React Router.



About BruhDive

Transforming Learning Through Intelligent Assessment

Our Mission

At BruhDive, we're revolutionizing skill assessment by combining advanced analytics with personalized learning. Our platform goes beyond traditional testing to provide actionable insights that help users identify weaknesses, optimize study time, and accelerate their learning journey.

1000+ Active Learners

50+ Skills Assessed

75% User Satisfaction

Why Choose BruhDive?

Intelligent Analytics

Our sophisticated algorithms analyze your performance across multiple dimensions, providing detailed insights into your strengths and areas for improvement.

Personalized Pathways

Get customized learning recommendations tailored to your unique skill profile and learning objectives.

Real-Time Feedback

Receive immediate evaluation of your solutions with actionable suggestions for improvement.

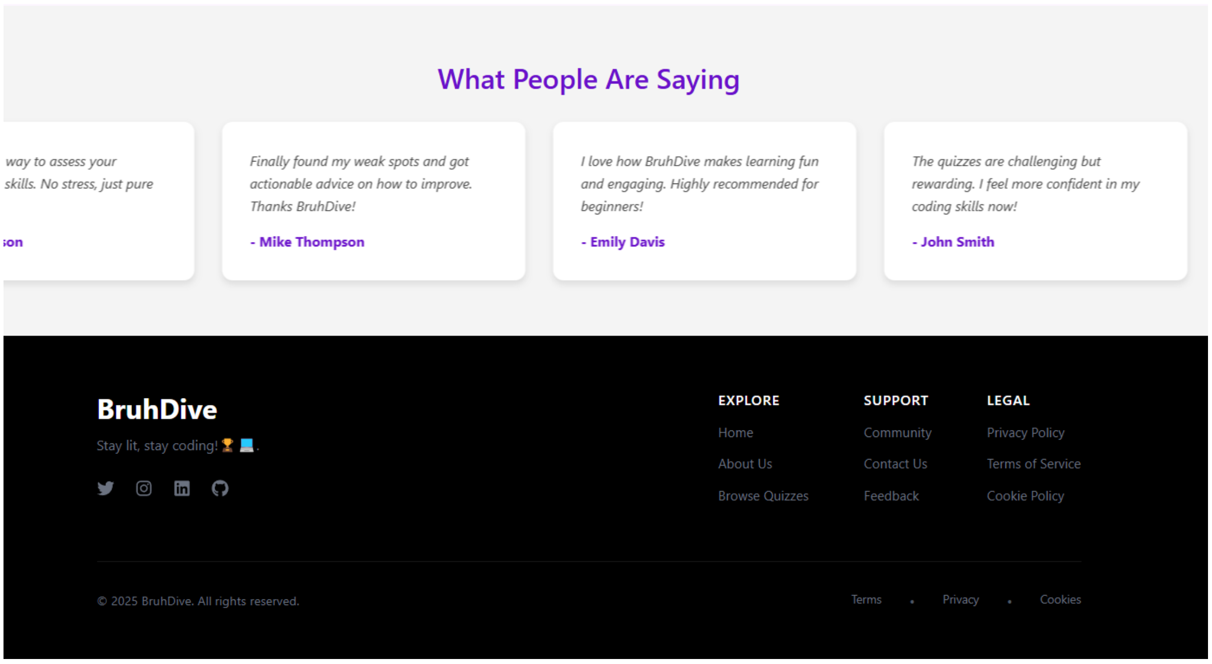
Comprehensive Skill Coverage

From fundamental concepts to advanced techniques, our assessments cover the full spectrum of technical skills.

Our Expertise

BruhDive was founded by a team of learning scientists, assessment specialists, and software engineers passionate about making skill development more efficient and effective. With decades of combined experience in education technology, we've created a platform that truly understands how people learn.

Ready to Transform Your Learning?


What People Are Saying

way to assess your skills. No stress, just pure

son

Finally found my weak spots and got actionable advice on how to improve. Thanks BruhDive!

- Mike Thompson

I love how BruhDive makes learning fun and engaging. Highly recommended for beginners!

- Emily Davis

The quizzes are challenging but rewarding. I feel more confident in my coding skills now!

- John Smith

BruhDive

Stay lit, stay coding! 🏆 💻.

EXPLORE	SUPPORT	LEGAL
Home	Community	Privacy Policy
About Us	Contact Us	Terms of Service
Browse Quizzes	Feedback	Cookie Policy

© 2025 BruhDive. All rights reserved.

Terms • Privacy • Cookies

Figure 12.2: About Page

➢ **<u>Quiz Page</u> (Language-Specific)**

<u>Description:</u>

Once a language quiz is started, the user sees one question at a time with options. The quiz is timer-independent and features 25 fixed questions per language. All questions are locally loaded and stored temporarily until submission.

File: Quiz.jsx (Quiz Page Entry Point)

```
import React, { useState } from "react";
import LanguageCard from "./LanguageCard";

const Quiz = () => {
  const [searchTerm, setSearchTerm] = useState("");
  const languages = [
    "Python", "Java", "C++", "JavaScript", "PHP",
    "SQL", "C", "C#", "Kotlin", "Test Yourself",
  ];

  const filteredLanguages = languages.filter((lang) =>
    lang.toLowerCase().includes(searchTerm.toLowerCase())
  );

  return (
    <div className="quiz-container">
      <h1>Master Your Coding Journey</h1>
      <p>Select a programming language or domain to begin your quiz.</p>

      <input
        type="text"
        placeholder="Search..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
      />

      <div className="language-grid">
        {filteredLanguages.map((lang, index) => (
          <LanguageCard key={index} language={lang} />
        ))}
      </div>
    </div>
  );
};

export default Quiz;
```

File: LanguageCard.jsx (Card Component for Each Quiz Option)

```
import React from "react";
import { useNavigate } from "react-router-dom";

const LanguageCard = ({ language }) => {
  const navigate = useNavigate();

  const icons = {
    python: "🐍", java: "☕", "c++": "🔧", javascript: "✨",
    php: "💻", sql: "📊", c: "⚙️", "c#": "🎮", kotlin: "🤖",
    path: "🛡️"
  };

  const descriptions = {
    Python: "Simple yet powerful. Ideal for AI, Data, Web Dev.",
    Java: "Enterprise-level object-oriented language.",
    "C++": "High-performance systems and game dev language.",
    JavaScript: "The king of web interactivity.",
    PHP: "Backend scripting for dynamic websites.",
    SQL: "Standard for querying and managing databases.",
    C: "The root of systems programming.",
    "C#": "Popular for Windows apps and game dev (Unity).",
    Kotlin: "Modern Android development language.",
    Path: "Not sure what to pick? Test Yourself to explore careers!",
  };

  const handleClick = () => {
    const path = language.toLowerCase().replace("#", "sharp");
    navigate(`/quiz/${path}`);
  };

  return (
    <div className="language-card">
      <div className="language-icon">{icons[language.toLowerCase()] || "💡"}</div>
      <h3>{language}</h3>
      <p>{descriptions[language] || "Discover your tech path!"}</p>
      <button onClick={handleClick}>Dive In! 🚀</button>
    </div>
  );
};
```

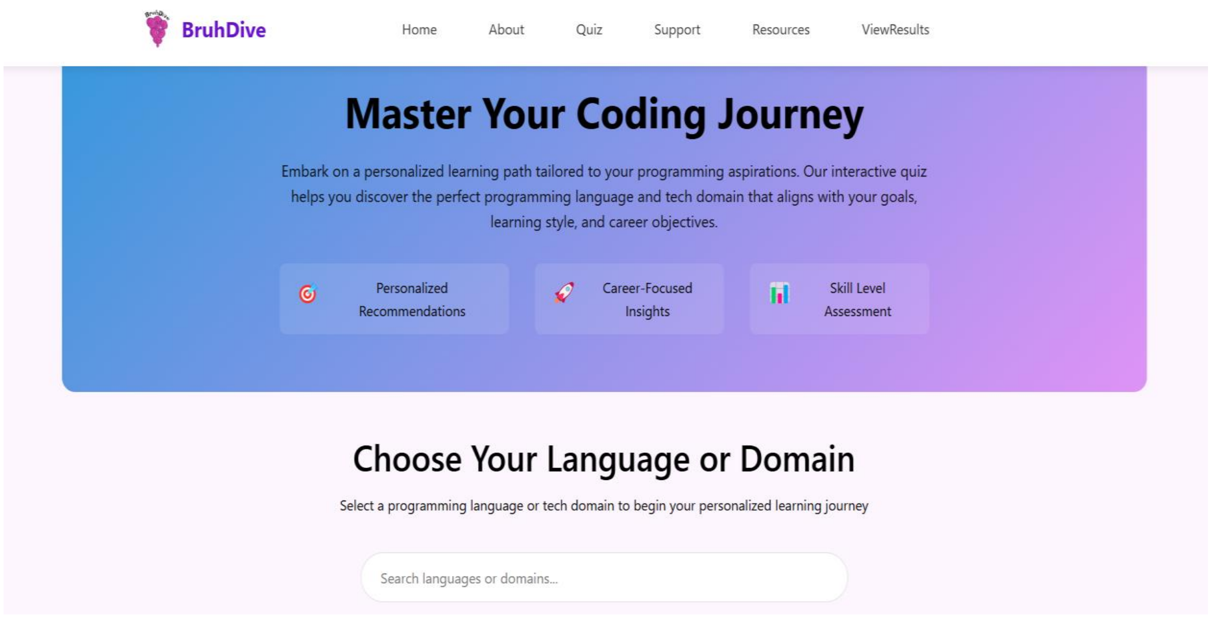
export default LanguageCard;

Description for Documentation:

The **Quiz Page** is the central hub for launching quizzes on different programming languages. Features include:

- A search bar to filter available quiz categories.
- Cards for each quiz type with icons and descriptions.
- A dynamic routing mechanism to navigate users to their selected quiz.

"Test Yourself" is a special card that loads domain-wise questions via API for **career guidance**.

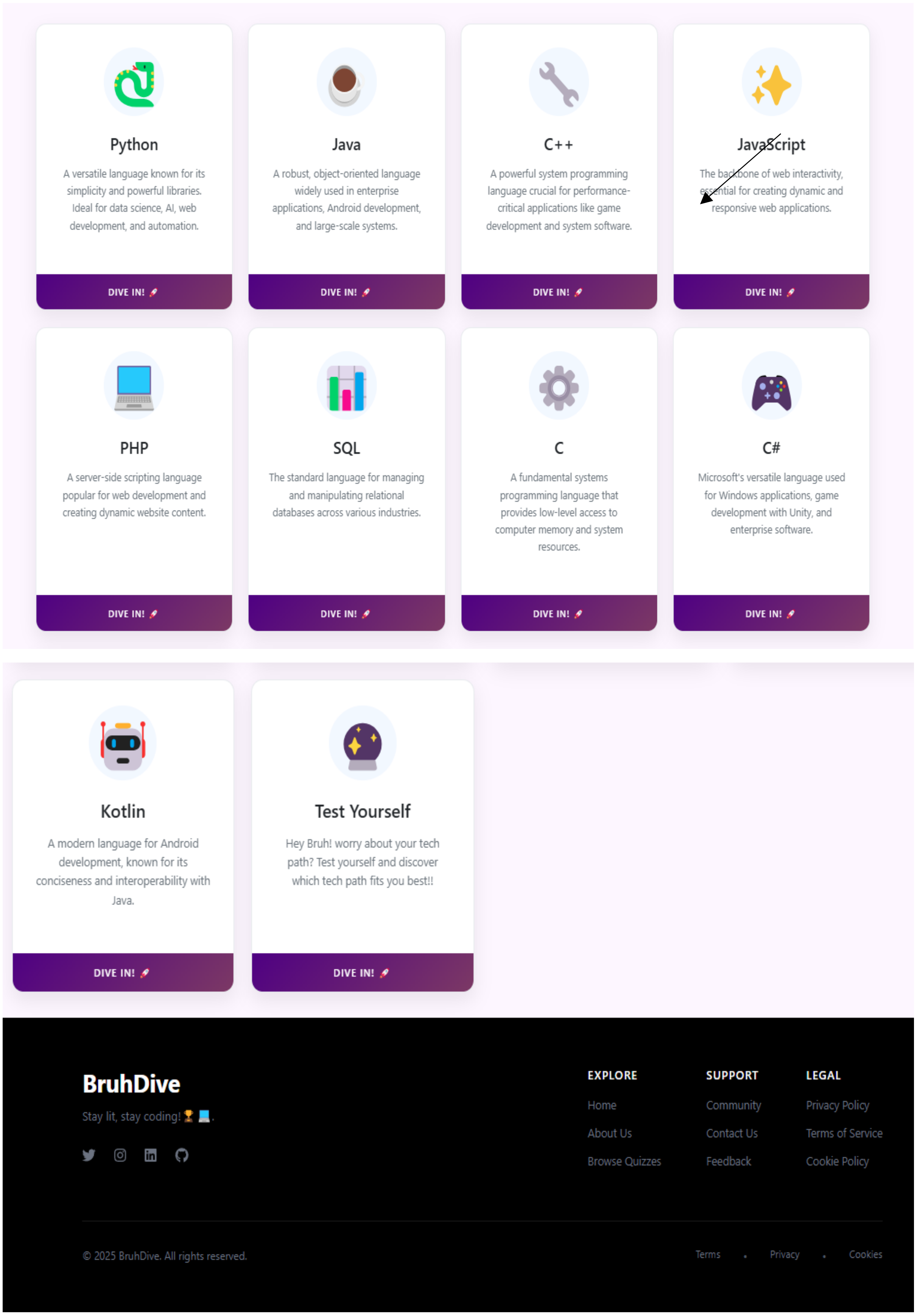


Master Your Coding Journey

Embark on a personalized learning path tailored to your programming aspirations. Our interactive quiz helps you discover the perfect programming language and tech domain that aligns with your goals, learning style, and career objectives.

Choose Your Language or Domain

Select a programming language or tech domain to begin your personalized learning journey



Python

A versatile language known for its simplicity and powerful libraries. Ideal for data science, AI, web development, and automation.

DIVE IN! 🚀

Java

A robust, object-oriented language widely used in enterprise applications, Android development, and large-scale systems.

DIVE IN! 🚀

C++

A powerful system programming language crucial for performance-critical applications like game development and system software.

DIVE IN! 🚀

JavaScript

The backbone of web interactivity, essential for creating dynamic and responsive web applications.

DIVE IN! 🚀

PHP

A server-side scripting language popular for web development and creating dynamic website content.

DIVE IN! 🚀

SQL

The standard language for managing and manipulating relational databases across various industries.

DIVE IN! 🚀

C

A fundamental systems programming language that provides low-level access to computer memory and system resources.

DIVE IN! 🚀

C#

Microsoft's versatile language used for Windows applications, game development with Unity, and enterprise software.

DIVE IN! 🚀

Kotlin

A modern language for Android development, known for its conciseness and interoperability with Java.

DIVE IN! 🚀

Test Yourself

Hey Bruh! worry about your tech path? Test yourself and discover which tech path fits you best!!

DIVE IN! 🚀

BruhDive

Stay lit, stay coding! 🏆 💻.

EXPLORE	SUPPORT	LEGAL
Home	Community	Privacy Policy
About Us	Contact Us	Terms of Service
Browse Quizzes	Feedback	Cookie Policy

© 2025 BruhDive. All rights reserved.

Terms • Privacy • Cookies

Figure 12.3: Quiz Page

Example Python Quiz:

Language-Specific Quiz Logic – Common to All Languages

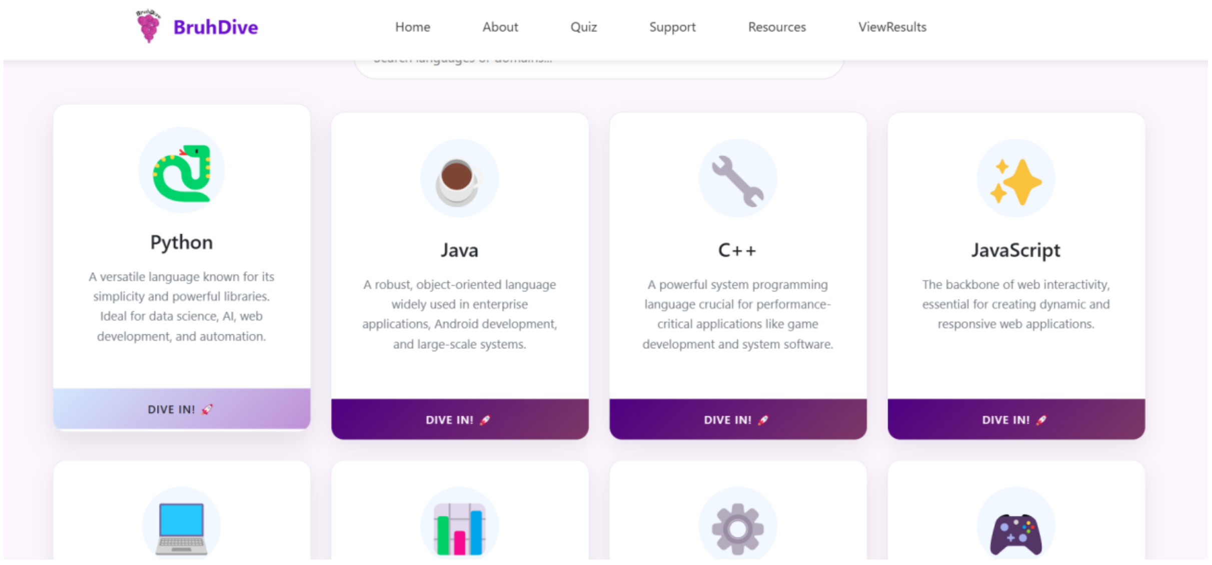
The same logic applies to Python, Java, JavaScript, PHP, etc. with only language-specific files and sets changing.

File: PythonQuiz.jsx *(Main quiz logic)*

```
import React, { useState, useEffect } from "react";
import Countdown from "./QuizComponents/PyCountdown";
import Question from "./QuizComponents/PyQuestion";
import Results from "./QuizComponents/PyResults";
import QuizNav from "./QuizComponents/PyQuizNav";
import { pythonQuestions } from "../../../data/PythonQns/pyQuestions";

const PythonQuiz = () => {
  const [quizStarted, setQuizStarted] = useState(false);
  const [quizFinished, setQuizFinished] = useState(false);
  const [questions, setQuestions] = useState([]);
  const [answers, setAnswers] = useState([]);
  const [currentIndex, setCurrentIndex] = useState(0);
  const [score, setScore] = useState(0);

  useEffect(() => {
    if (quizStarted) {
      setQuestions(pythonQuestions);
      setAnswers(Array(pythonQuestions.length).fill(null));
    }
  }, [quizStarted]);

  const handleStart = () => setQuizStarted(true);

  const handleAnswer = (option) => {
    const updated = [...answers];
    updated[currentIndex] = option;
    setAnswers(updated);
  };
```

```
const calculateScore = () => {
  const correct = questions.filter(
    (q, i) => answers[i] === q.correctAnswer
  ).length;
  setScore(correct);
  setQuizFinished(true);
};

if (!quizStarted) return <Countdown onStart={handleStart} />;
if (quizFinished)
  return (
    <Results
      score={score}
      totalQuestions={questions.length}
      userAnswers={answers}
      questionData={questions}
      onRestart={() => window.location.reload()}
    />
  );
return (
  <div>
    <QuizNav currentIndex={currentIndex} totalQuestions={questions.length} />
    <Question
      question={questions[currentIndex]}
      selectedOption={answers[currentIndex]}
      onOptionChange={handleAnswer}
      currentIndex={currentIndex}
      totalQuestions={questions.length}
      onNext={() => setCurrentIndex((prev) => prev + 1)}
      onPrevious={() => setCurrentIndex((prev) => prev - 1)}
      onSubmit={calculateScore}
    />
  </div>
);
```

```
};

export default PythonQuiz;
```



Python

A versatile language known for its simplicity and powerful libraries. Ideal for data science, AI, web development, and automation.

DIVE IN!

Java

A robust, object-oriented language widely used in enterprise applications, Android development, and large-scale systems.

DIVE IN!

C++

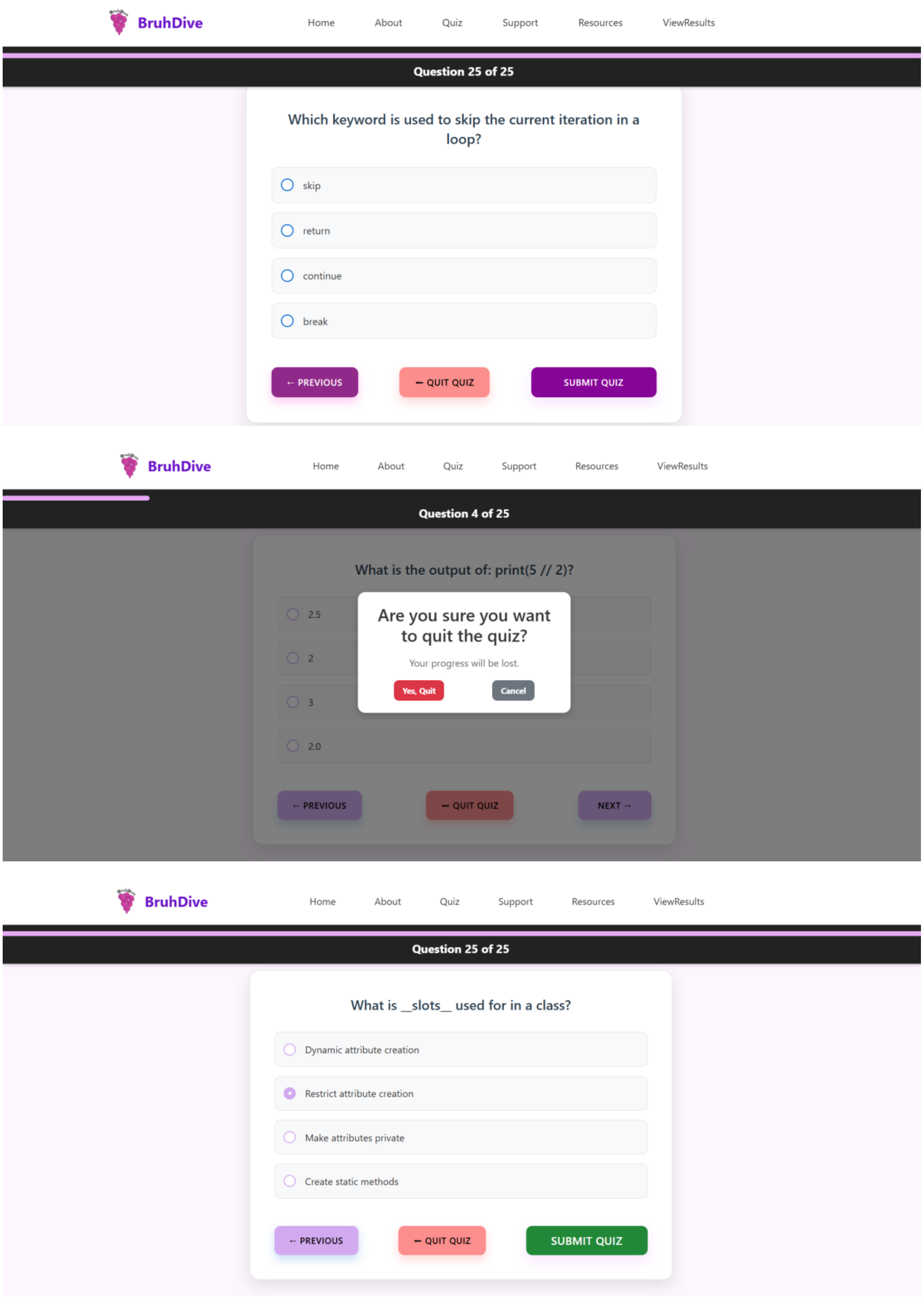
A powerful system programming language crucial for performance-critical applications like game development and system software.

DIVE IN!

JavaScript

The backbone of web interactivity, essential for creating dynamic and responsive web applications.

DIVE IN!



Question 25 of 25

Which keyword is used to skip the current iteration in a loop?

- skip
- return
- continue
- break

Question 4 of 25

What is the output of: print(5 // 2)?

- 2.5
- 2
- 3
- 2.0

Are you sure you want to quit the quiz?

Your progress will be lost.

Question 25 of 25

What is __slots__ used for in a class?

- Dynamic attribute creation
- [✓ Restrict attribute creation]
- Make attributes private
- Create static methods

Figure 12.4: Python Page (Example)

➢ Test Yourself (Career Quiz)

Description:

This is a special quiz card that dynamically fetches questions from an AI-based API. It evaluates user responses across different domains like Web Development, Cybersecurity, DevOps, etc., and provides personalized career suggestions based on performance.

If the API fails, the fallback system loads a set of hardcoded questions to ensure uninterrupted functionality.

File: QuizPage.jsx *(Career Guidance Logic)*

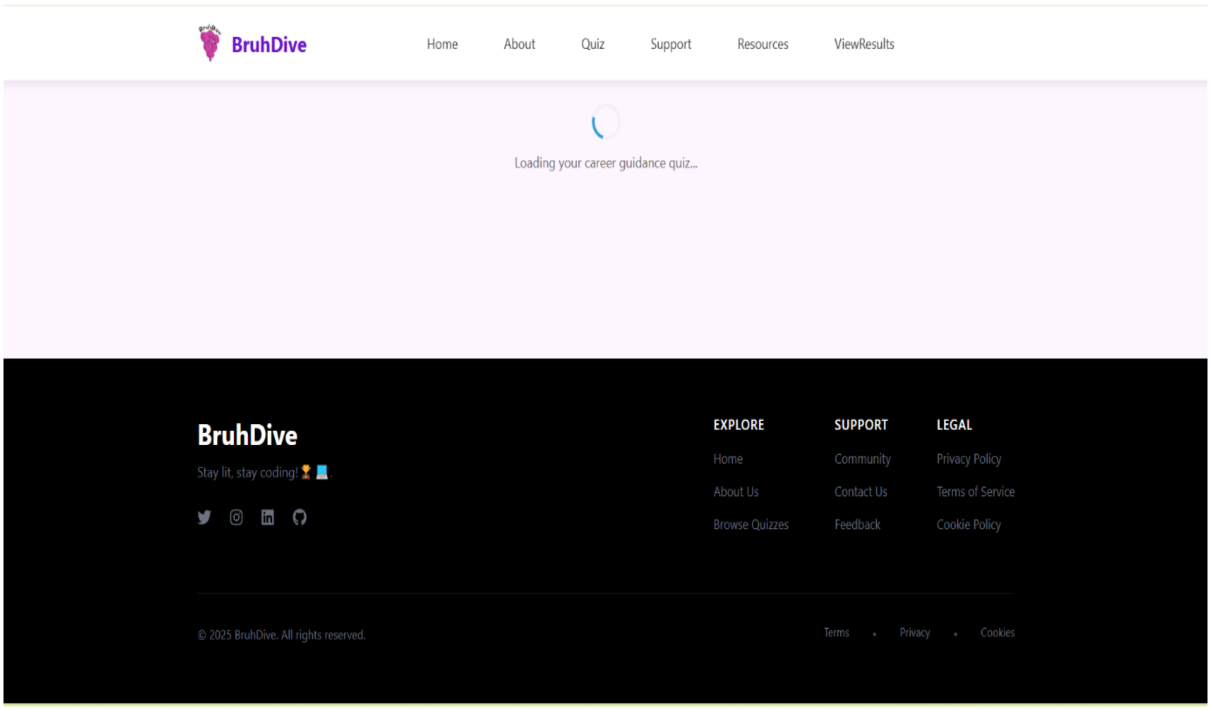
```
import { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import Confetti from "react-confetti";
import { fetchQuiz } from "../../../utils/fetchQuiz";

function QuizPage() {
  const navigate = useNavigate();
  const [questions, setQuestions] = useState([]);
  const [currentQ, setCurrentQ] = useState(0);
  const [answers, setAnswers] = useState({});
  const [completed, setCompleted] = useState(false);
  const [recommendations, setRecommendations] = useState([]);

  useEffect(() => {
    fetchQuiz().then((data) => setQuestions(data));
  }, []);

  const handleAnswer = (answer) => {
    setAnswers({ ...answers, [currentQ]: answer });
  };

  const finishQuiz = () => {
    setCompleted(true);
    fetch("/api/career-recommendations", {
```

```
    method: "POST",
    body: JSON.stringify({ answers }),
    headers: { "Content-Type": "application/json" },
  })
    .then((res) => res.json())
    .then((data) => setRecommendations(data))
    .catch(() => {
      // fallback recs
      setRecommendations([{ title: "Tech Explorer", icon: "🔍" }]);
    });
};

if (completed) {
  return (
    <div>
      <Confetti />
      <h1>Career Suggestions</h1>
      {recommendations.map((rec, i) => (
        <div key={i}>
          <h2>{rec.icon} {rec.title}</h2>
          <p>{rec.description}</p>
        </div>
      ))}
      <button onClick={() => navigate("/resources")}>Resources</button>
    </div>
  );
}

const q = questions[currentQ];

return (
  <div>
    <h1>Career Guidance Quiz</h1>
```

```
      <p>Question {currentQ + 1} of {questions.length}</p>
      <h2>{q?.question}</h2>
      <div>
        {q?.options?.map((opt, idx) => (
          <button key={idx} onClick={() => handleAnswer(opt.charAt(0))}>
            {opt}
          </button>
        ))}
      </div>
      <button disabled={currentQ === 0} onClick={() => setCurrentQ(currentQ - 1)}>
        Previous
      </button>
      <button
        disabled={!answers[currentQ]}
        onClick={
          currentQ === questions.length - 1
            ? finishQuiz
            : () => setCurrentQ(currentQ + 1)
        }
      >
        {currentQ === questions.length - 1 ? "Finish" : "Next"}
      </button>
    </div>
  );
}

export default QuizPage;
```

What This Does:

- Dynamically fetches **domain-wise quiz questions**
- Tracks answers and progress
- On completion, shows **personalized career recommendations**
- Uses a fallback suggestion if the API fails
- Offers navigation to resources for further learning



BruhDive

Home About Quiz Support Resources ViewResults

Loading your career guidance quiz...

BruhDive

Stay lit, stay coding! 🏆 📘

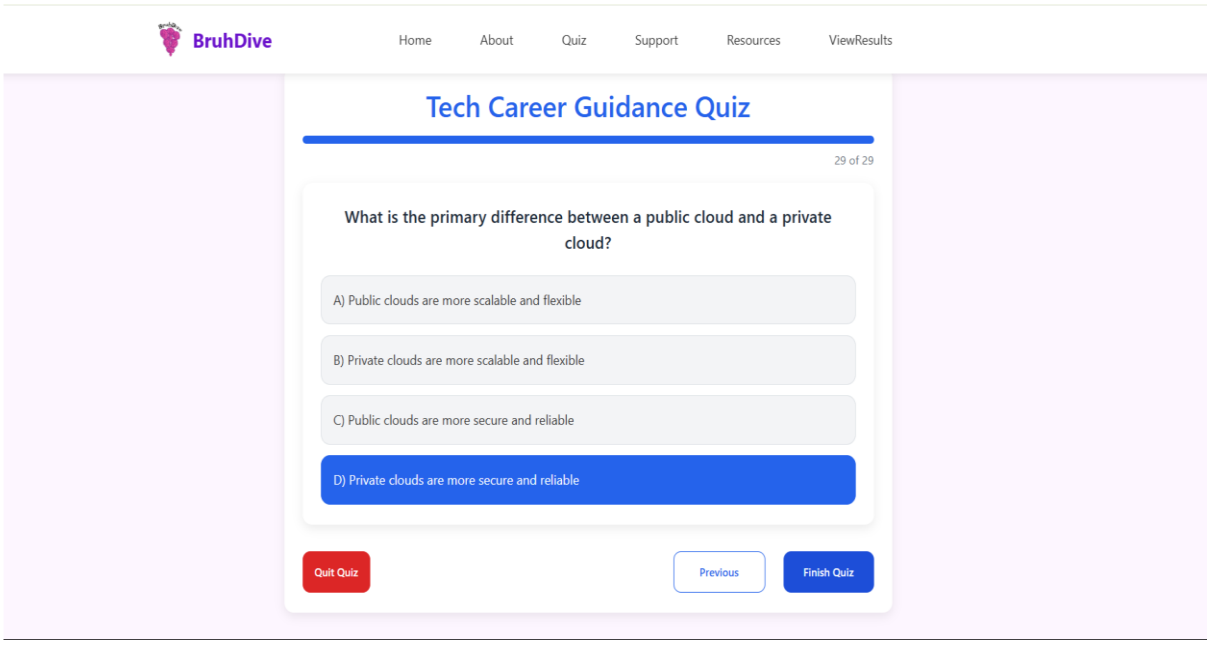
EXPLORE: Home, About Us, Browse Quizzes

SUPPORT: Community, Contact Us, Feedback

LEGAL: Privacy Policy, Terms of Service, Cookie Policy

© 2025 BruhDive. All rights reserved.

Terms • Privacy • Cookies



BruhDive

Home About Quiz Support Resources ViewResults

Tech Career Guidance Quiz

29 of 29

What is the primary difference between a public cloud and a private cloud?

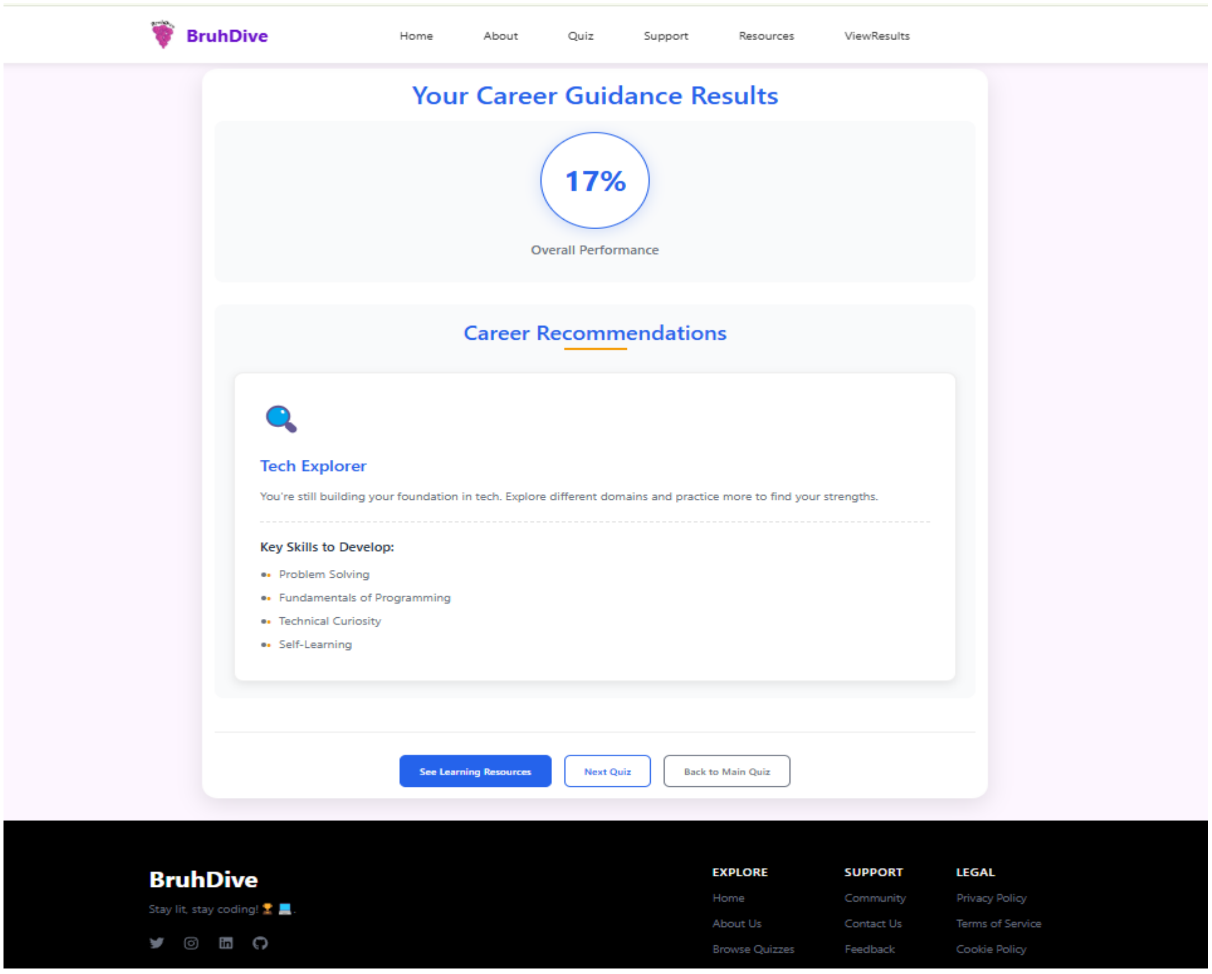
A) Public clouds are more scalable and flexible

B) Private clouds are more scalable and flexible

C) Public clouds are more secure and reliable

[✓ D) Private clouds are more secure and reliable]

Quit Quiz | Previous | Finish Quiz



Your Career Guidance Results

17%

Overall Performance

Career Recommendations

Tech Explorer

You're still building your foundation in tech. Explore different domains and practice more to find your strengths.

Key Skills to Develop:

- Problem Solving
- Fundamentals of Programming
- Technical Curiosity
- Self-Learning

Figure 12.5: Test Yourself Page

➢ Result Page

Description:

After quiz submission, result page is shown where user can view:

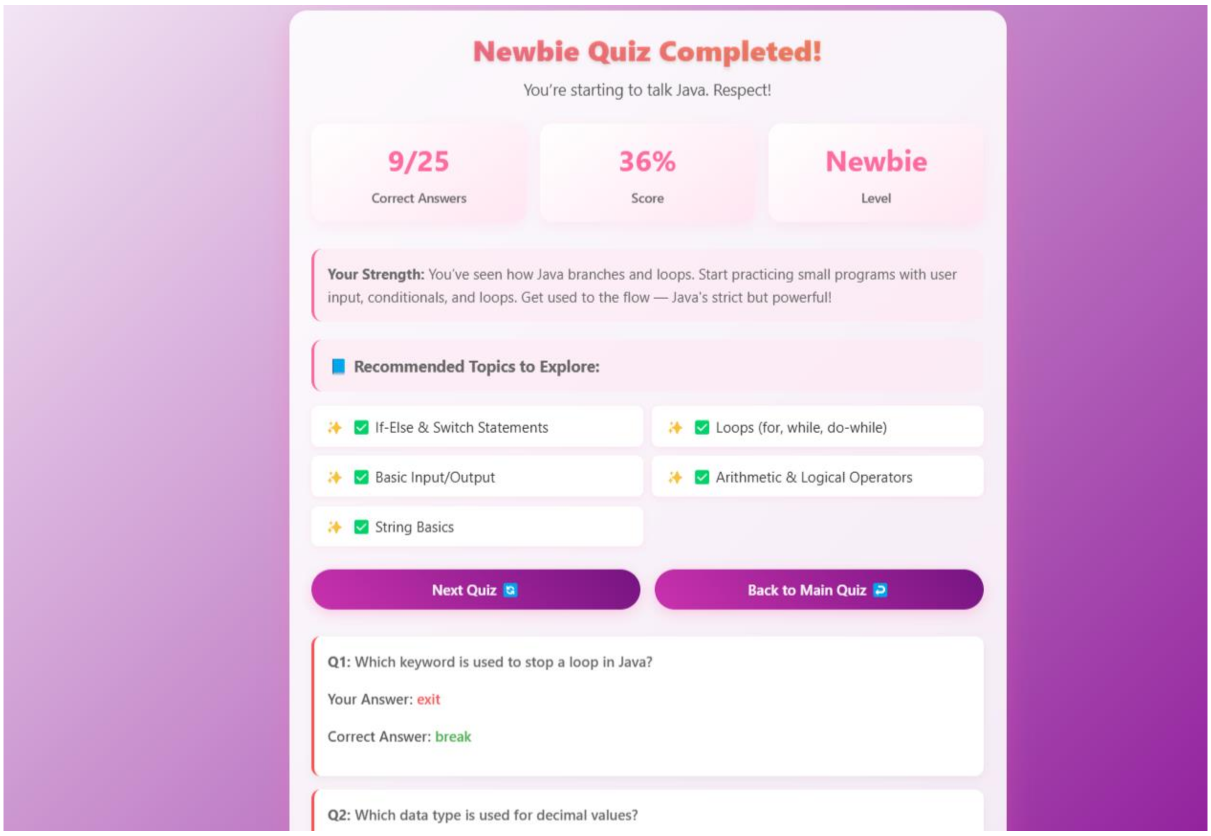
- Total score and percentage
- Performance level (e.g., Beginner, Advanced, Prodigy)
- Strengths across topics
- Recommended learning areas or career paths

File: Results.jsx *(Quiz Summary & Feedback)*

```
import React, { useEffect, useRef, useState } from "react";
import { useNavigate } from "react-router-dom";
import Confetti from "react-confetti";

const Results = ({ score, totalQuestions, onRestart, userAnswers, questionData }) => {
  const navigate = useNavigate();
  const hasSaved = useRef(false);
  const percentage = Math.round((score / totalQuestions) * 100);

  const getPerformanceLevel = () => {
    if (score <= 4) return { level: "Noob Mode", message: "Keep going!", color: "#6c757d" };
    if (score <= 9) return { level: "Newbie", message: "You're getting there!", color: "#fd7e14"
};
    if (score <= 13) return { level: "Beginner", message: "Solid base!", color: "#ffc107" };
    if (score <= 17) return { level: "Enthusiast", message: "You’re doing great!", color:
"#17a2b8" };
    if (score <= 20) return { level: "Intermediate", message: "Almost pro!", color: "#007bff" };
    if (score <= 23) return { level: "Advanced", message: "Big brain moves!", color: "#28a745"
};
    return { level: "Code Master", message: "Certified legend!", color: "#6610f2" };
  };

  const performance = getPerformanceLevel();

  useEffect(() => {
if (!hasSaved.current) {
      const resultData = {
        language: "Python", // dynamic if needed
        score,
        total: totalQuestions,
        percentage,
        level: performance.level,
        timestamp: Date.now(),
      };
      const history = JSON.parse(localStorage.getItem("quizResults")) || [];
      history.push(resultData);
      localStorage.setItem("quizResults", JSON.stringify(history));
      hasSaved.current = true;
    }
  }, [score, totalQuestions, percentage, performance]);

  return (
    <div className="results-container">
      <Confetti numberOfPieces={150} recycle={false} />
      <h1>{performance.level} Quiz Completed!</h1>
      <p>{performance.message}</p>

      <div className="result-metrics">
        <p>Score: {score} / {totalQuestions} ({percentage}%)</p>
        <p>Level: {performance.level}</p>
      </div>

      <button onClick={onRestart}>Next Quiz 🔄</button>
      <button onClick={() => navigate("/quiz")}>Back to Main Quiz ↩</button>
    </div>
  );
};

export default Results;
```

What it Covers:

- Displays score, percentage, and performance level.
- Saves result to localStorage.
- Shows confetti on completion.
- Offers navigation to retake quiz or go back.



Newbie Quiz Completed!

You're starting to talk Java. Respect!

9/25
Correct Answers

36%
Score

Newbie
Level

Your Strength: You've seen how Java branches and loops. Start practicing small programs with user input, conditionals, and loops. Get used to the flow — Java's strict but powerful!

📘 **Recommended Topics to Explore:**

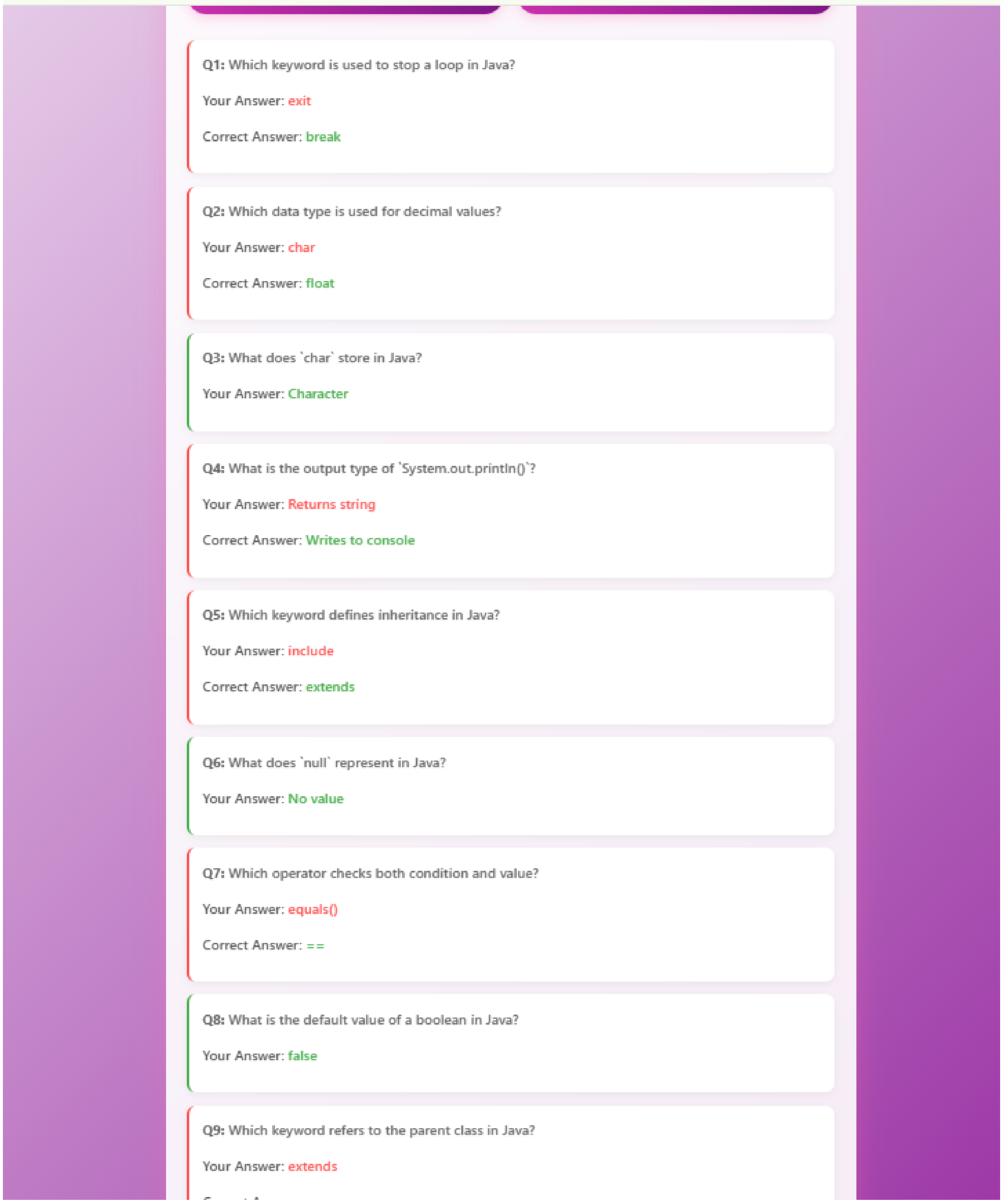
- ✨ ✅ If-Else & Switch Statements
- ✨ ✅ Loops (for, while, do-while)
- ✨ ✅ Basic Input/Output
- ✨ ✅ Arithmetic & Logical Operators
- ✨ ✅ String Basics

Q1: Which keyword is used to stop a loop in Java?

Your Answer: exit

Correct Answer: break

Q2: Which data type is used for decimal values?



Q1: Which keyword is used to stop a loop in Java?

Your Answer: exit

Correct Answer: break

Q2: Which data type is used for decimal values?

Your Answer: char

Correct Answer: float

Q3: What does `char` store in Java?

Your Answer: Character

Q4: What is the output type of `System.out.println()`?

Your Answer: Returns string

Correct Answer: Writes to console

Q5: Which keyword defines inheritance in Java?

Your Answer: include

Correct Answer: extends

Q6: What does `null` represent in Java?

Your Answer: No value

Q7: Which operator checks both condition and value?

Your Answer: equals()

Correct Answer: ==

Q8: What is the default value of a boolean in Java?

Your Answer: false

Q9: Which keyword refers to the parent class in Java?

Your Answer: extends

Figure 12.6: Result Page

➢ **<u>View Results (History)</u>**

<u>Description:</u>

This page displays the user's past quiz attempts using data saved in localStorage. Users can revisit past performances.

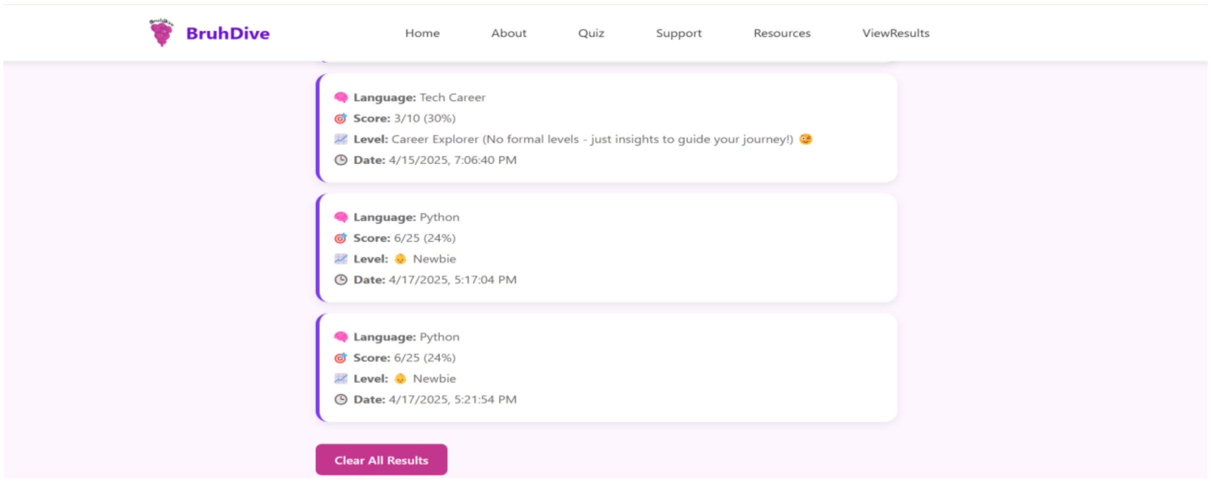
File: ViewResults.jsx

```
   import React, { useEffect, useState } from "react";
   import "./ViewResults.css"; // Custom styling

const ViewResults = () => {
    const [results, setResults] = useState([]);

    useEffect(() => {
     const stored = JSON.parse(localStorage.getItem("quizResults")) || [];
     const parsed = stored.map((res) => ({
      ...res,
      percentage: Math.round((res.score / (res.total || 25)) * 100),
      date: new Date(res.timestamp || Date.now()).toLocaleString(),
     }));
     setResults(parsed);
    }, []);

    const clearResults = () => {
     localStorage.removeItem("quizResults");
     setResults([]);
    };

    return (
     <div className="view-results-container">
      <h1>📜 Your Quiz History</h1>
      {results.length === 0 ? (
       <p>No past results found. Go dive into some quizzes!</p>
      ) : (
       <>
        {results.map((res, i) => (
         <div key={i} className="result-card">
          <p><strong>□ Language:</strong> {res.language}</p>
          <p><strong>🎯          Score:</strong>          {res.score}/{res.total}
   ({res.percentage}%)</p>
```

```
          <p><strong>📈 Level:</strong> {res.level}</p>
          <p><strong>🕒 Date:</strong> {res.date}</p>
        </div>
      ))}
      <button onClick={clearResults} className="clear-btn">
        Clear All Results
      </button>
    </>
  )}
 </div>
);
};

export default ViewResults;
```

What It Covers:

- Fetches past quiz results from localStorage
- Displays:
    - Language
    - Score & %
    - Level
    - Timestamp
- Includes a **Clear All** button



Language: Tech Career
Score: 3/10 (30%)
Level: Career Explorer (No formal levels - just insights to guide your journey!) 😉
Date: 4/15/2025, 7:06:40 PM

Language: Python
Score: 6/25 (24%)
Level: 👶 Newbie
Date: 4/17/2025, 5:17:04 PM

Language: Python
Score: 6/25 (24%)
Level: 👶 Newbie
Date: 4/17/2025, 5:21:54 PM

Figure 12.7: View results Page

➢ **Resources Page**

Description:

The Resources Page provides categorized learning materials and useful links including:

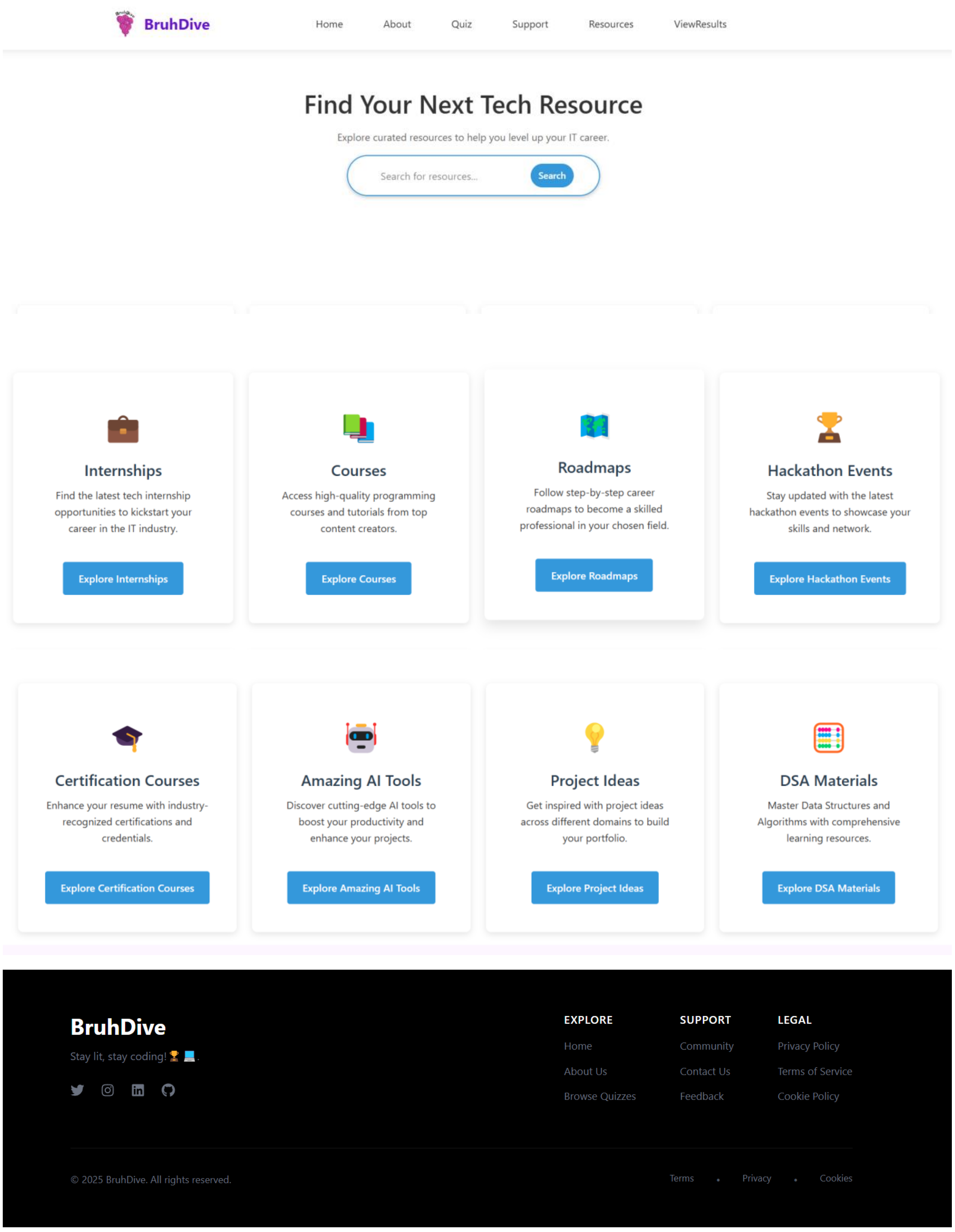
- Roadmaps for Web Dev, DSA, etc.
- Internships and Certifications
- Online Courses
- Tools and Projects
- Events and Hackathons

File: ResourcesPage.jsx

```
import React, { useState } from "react";
import { Link } from "react-router-dom";
import "./resources.css";

const resourceData = [
  { icon: "💼", title: "Internships", link: "/internships" },
  { icon: "📚", title: "Courses", link: "/courses" },
  { icon: "🗺️", title: "Roadmaps", link: "/roadmaps" },
  { icon: "🏆", title: "Hackathons", link: "/hackathons" },
  { icon: "🎓", title: "Certifications", link: "/certifications" },
  { icon: "🤖", title: "AI Tools", link: "/ai-tools" },
  { icon: "💡", title: "Project Ideas", link: "/project-ideas" },
  { icon: "□", title: "DSA Materials", link: "/dsa-materials" },
];

const ResourcesPage = () => {
  const [search, setSearch] = useState("");
  const filtered = resourceData.filter((res) =>
    res.title.toLowerCase().includes(search.toLowerCase())
  );

  return (
    <div className="resources-page">
      <h1>Explore Resources 🔎</h1>
      <input
        className="resource-search"
        type="text"
        placeholder="Search Resources..."
        value={search}
        onChange={(e) => setSearch(e.target.value)}
      />
      <div className="resources-grid">
        {filtered.length > 0 ? (
          filtered.map((res, i) => (
            <Link to={res.link} className="resource-card" key={i}>
              <div className="icon">{res.icon}</div>
              <h3>{res.title}</h3>
            </Link>
          ))
        ) : (
          <p>❌ No matching resources found.</p>
        )}
      </div>
    </div>
  );
};

export default ResourcesPage;
```

What It Does:

- Displays **resource cards** with emojis + links
- Includes **search filter**
- Uses <Link> for routing (internal navigation)
- Minimal structure for clean styling


BruhDive

Home About Quiz Support Resources ViewResults

Find Your Next Tech Resource

Explore curated resources to help you level up your IT career.

Search for resources... Search

Internships

Find the latest tech internship opportunities to kickstart your career in the IT industry.

Explore Internships

Courses

Access high-quality programming courses and tutorials from top content creators.

Explore Courses

Roadmaps

Follow step-by-step career roadmaps to become a skilled professional in your chosen field.

Explore Roadmaps

Hackathon Events

Stay updated with the latest hackathon events to showcase your skills and network.

Explore Hackathon Events

Certification Courses

Enhance your resume with industry-recognized certifications and credentials.

Explore Certification Courses

Amazing AI Tools

Discover cutting-edge AI tools to boost your productivity and enhance your projects.

Explore Amazing AI Tools

Project Ideas

Get inspired with project ideas across different domains to build your portfolio.

Explore Project Ideas

DSA Materials

Master Data Structures and Algorithms with comprehensive learning resources.

Explore DSA Materials

BruhDive

Stay lit, stay coding! 🏆 💻.

EXPLORE: Home, About Us, Browse Quizzes

SUPPORT: Community, Contact Us, Feedback

LEGAL: Privacy Policy, Terms of Service, Cookie Policy

© 2025 BruhDive. All rights reserved.

Terms • Privacy • Cookies

Figure 12.8: Resources Page

➢ <u>Support Page</u> (Feedback & Contact)

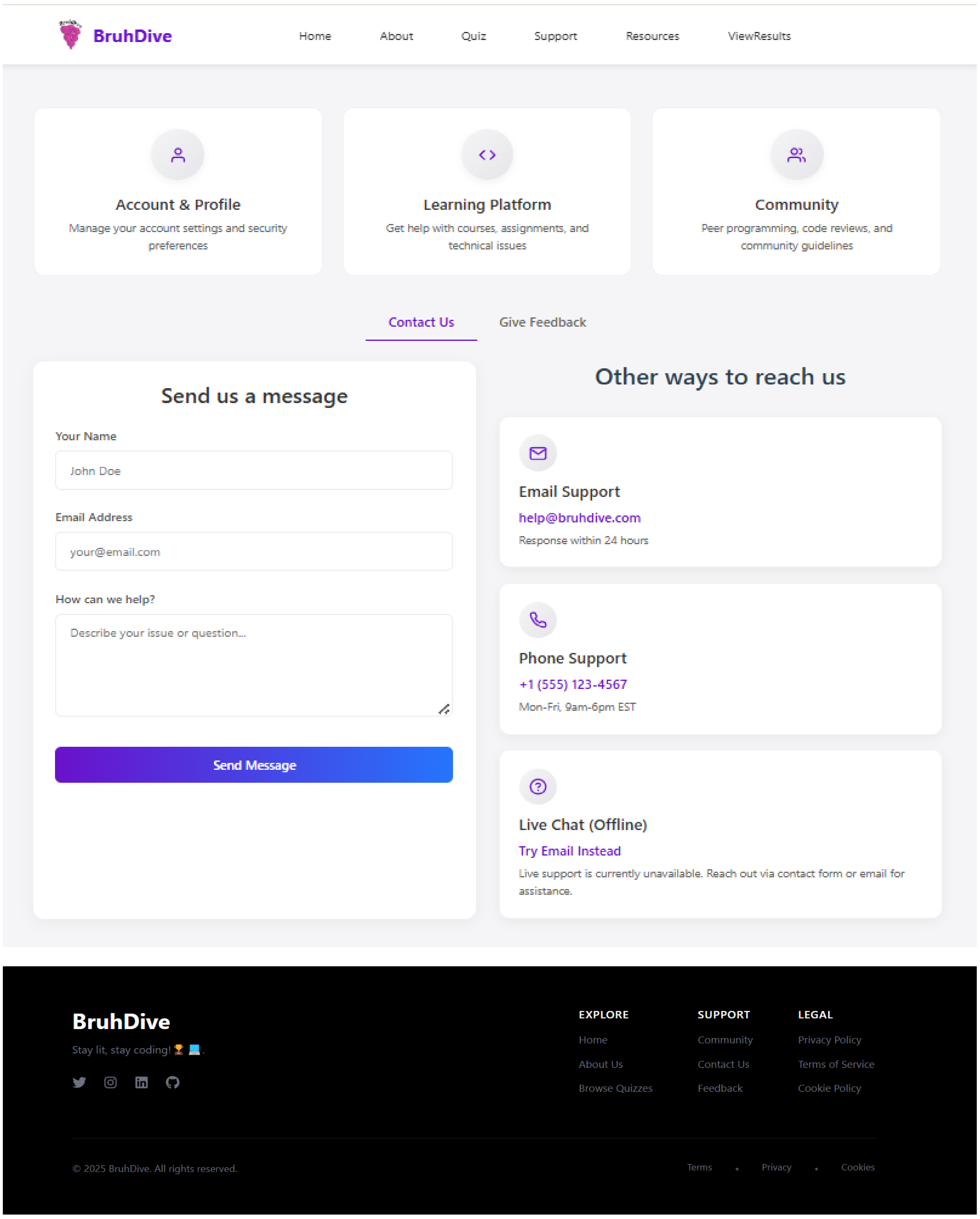
Description:

The Support Page allows users to reach out for help or provide feedback. It contains a contact form and a simple feedback submission area.

File: ContactPage.jsx

```
import React, { useState, useEffect } from "react";
import emailjs from "@emailjs/browser";
import { FiMail, FiPhone, FiUser, FiCode, FiUsers } from "react-icons/fi";
import { FaStar, FaRegStar } from "react-icons/fa";
import "./ContactPage.css";

const ContactPage = () => {
  const [form, setForm] = useState({ name: "", email: "", message: "", feedback: "" });
  const [rating, setRating] = useState(0);
  const [tab, setTab] = useState("contact");
  const [loading, setLoading] = useState(false);
  const [popup, setPopup] = useState(null);

  useEffect(() => {
    emailjs.init("YOUR_PUBLIC_KEY");
  }, []);

  const show = (msg, type = "success") => {
    setPopup({ msg, type });
    setTimeout(() => setPopup(null), 4000);
  };

  const handleChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!form.name || !form.email || (tab === "contact" && !form.message)) {
      show("Please fill all required fields", "error");
      return;
    }

    setLoading(true);
    try {
      const params = { ...form, to_email: form.email };
      await emailjs.send("YOUR_SERVICE_ID", "YOUR_TEMPLATE_ID", params);
      show(tab === "contact" ? "Message sent!" : "Feedback submitted!");
      setForm({ name: "", email: "", message: "", feedback: "" });
      setRating(0);
    } catch {
      show("Something went wrong. Try again.", "error");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="contact-page">
      {popup && <div className={`popup ${popup.type}`}>{popup.msg}</div>}

      <div className="support-tabs">
        <button className={tab === "contact" ? "active" : ""} onClick={() =>
setTab("contact")}>
          Contact Us
        </button>
        <button className={tab === "feedback" ? "active" : ""} onClick={() =>
setTab("feedback")}>
          Feedback
        </button>
      </div>

      <form onSubmit={handleSubmit} className="contact-form">
        <input name="name" value={form.name} onChange={handleChange}
placeholder="Your Name" required />
        <input type="email" name="email" value={form.email} onChange={handleChange}
placeholder="Your Email" required />

        {tab === "contact" ? (
          <textarea
            name="message"
            value={form.message}
            onChange={handleChange}
            placeholder="How can we help you?"
            required
          />
        ) : (
          <>
            <div className="rating">
              {[1, 2, 3, 4, 5].map((star) => (
                <span key={star} onClick={() => setRating(star)}>
                  {rating >= star ? <FaStar /> : <FaRegStar />}
                </span>
              ))}
            </div>
            <textarea
              name="feedback"
              value={form.feedback}
              onChange={handleChange}
              placeholder="Your feedback (optional)"
            />
          </>
        )}

        <button type="submit" disabled={loading}>
          {loading ? "Sending..." : tab === "contact" ? "Send Message" : "Submit Feedback"}
        </button>
```

```
      </form>

      <div className="support-info">
        <div><FiMail /> help@bruhdive.com</div>
        <div><FiPhone /> +1 (555) 123-4567</div>
      </div>

      <div className="support-categories">
        {[FiUser, FiCode, FiUsers].map((Icon, i) => (
          <div key={i} className="support-box">
            <Icon size={20} />
            <span>{["Account", "Learning", "Community"][i]}</span>
          </div>
        ))}
      </div>
    </div>
  );
};

export default ContactPage;
```



Account & Profile

Manage your account settings and security preferences

Learning Platform

Get help with courses, assignments, and technical issues

Community

Peer programming, code reviews, and community guidelines

Send us a message

Your Name: John Doe

Email Address: your@email.com

How can we help?: Describe your issue or question...

Other ways to reach us

Email Support
help@bruhdive.com
Response within 24 hours

Phone Support
+1 (555) 123-4567
Mon-Fri, 9am-6pm EST

Live Chat (Offline)
Try Email Instead
Live support is currently unavailable. Reach out via contact form or email for assistance.

BruhDive
Stay lit, stay coding! 🏆 💻.

© 2025 BruhDive. All rights reserved.

Figure 12.9: Support Page

13. TESTING

The testing process was conducted using Unit Testing, System Testing & End-to-End (E2E)Testing. Each phase of testing ensured that the BruhDive platform met its functional and non-functional requirements across all modules, from quiz handling to results visualization and user interaction.

13.1. UNIT TESTING

Unit Testing is a crucial testing phase that focuses on verifying individual components or functions of a software application in isolation. Each unit, whether a function, method, or component, is tested to ensure it performs as expected before being integrated into the larger system. By performing unit tests, developers can isolate bugs at the earliest stage of development, reducing the risk of errors and making maintenance easier.

Testing Method:

White Box Testing: This testing method involves testing internal structures or workings of an application, typically by looking at the code itself. Developers test individual units of code to ensure they behave correctly when isolated.

When to Perform:

Unit Testing is the first level of testing in the software development lifecycle. It is performed before integration testing to validate that each unit is functioning independently as expected.

Unit Testing Focus for BruhDive:

For **BruhDive**, unit tests would focus on:

- **React Components**: Ensuring that each React component renders correctly and handles user inputs as expected.
- **Utility Functions**: Testing functions used across the app, such as those for handling quiz results, score calculations, and data formatting.
- **API Call Functions**: Testing how the frontend interacts with the backend API, including fallback behavior when API requests fail.

Tasks:

1. **Prepare, Review, and Finalize Unit Test Plans**:
    - Created a detailed plan for testing each function and component in BruhDive. The plan specifies the tests for each module, what data will be used for testing, and the expected outcomes.
2. **Write, Review, and Finalize Unit Test Cases/Scripts**:
    - **Test Cases for React Components**: Write test cases for each React component. For example, testing whether the **QuizPage** component correctly displays questions, whether the **ResultPage** shows the score, level, and recommendations correctly, and whether the **SupportPage** form handles input properly.
    - **Test Cases for Utility Functions**: This includes testing functions that calculate scores, evaluate levels, handle API fallbacks, and store results in **localStorage**.
    - **Test Cases for API Calls**: Testing how the frontend interacts with the backend API to fetch quiz questions or career guidance, and ensuring that it switches to fallback data if the API call fails.
3. **Perform Unit Testing on Each Module**:
    - **Quiz Flow**: Ensuring that individual functions within the quiz (e.g., validating answers, tracking the current question, updating the score) work in isolation.
    - **Result Calculation**: Ensuring the functions that calculate the user’s quiz results, such as determining the score and assigning a level, work correctly based on the quiz answers.
    - **Career Guidance**: Testing the **Test Yourself** quiz's ability to generate career suggestions based on user performance.

Benefits of Unit Testing in BruhDive:

- **Improved Code Confidence**: Ensures that each component of BruhDive works correctly in isolation, making developers more confident when making changes.

- **Early Bug Detection**: Unit tests allow developers to catch bugs early, making the debugging process less expensive and time-consuming.
- **Better Maintainability**: When a bug is discovered, unit tests help identify the problematic area of code quickly, reducing the time it takes to fix issues.

13.2. SYSTEM TESTING

System Testing involves testing the entire BruhDive application as a whole. It ensures that all the components work together as expected and meet the defined requirements. System testing verifies that BruhDive performs all of its intended functions and behaves as a cohesive system.

Objective:

To ensure that **BruhDive** functions as a complete system, covering:

- All pages (e.g., Homepage, Quiz Pages, Result Pages, Support, and Resources)
- All quiz flows (including different programming languages, career guidance quiz, result calculations)
- The interaction between the client-side frontend and backend API, including fallback mechanisms.

Modules Tested:

1. **Homepage Navigation**:

   Testing that the homepage loads correctly and navigation between pages (e.g., quiz pages, result page, support page) is seamless.
2. **Quiz Component (for Multiple Languages)**:
   Ensuring that the quizzes for different programming languages (Python, JavaScript, Java, etc.) load properly, and that questions, answers, and navigation work as expected.
3. **"Test Yourself" AI-Powered Quiz**:

   Testing the functionality of the AI-generated career quiz, including the ability to fetch questions, process answers, and provide career recommendations based on performance.
4. **Result Page (Score, Level, Strengths, Recommendations)**:

Verifying that after a quiz is completed, the result page correctly displays the user's score, level, strengths, and suggested topics based on performance.

5. **Support & Contact Page**:

   Verifying that the support page functions properly, allowing users to submit feedback or contact the support team.

6. **Resources Page**:

   Ensuring that the resources page loads correctly, with links to learning materials, internships, roadmaps, etc.

Testing Method:

System testing is typically done in a **black-box** testing approach, focusing on the external behavior of the application rather than the internal workings. This approach helps ensure that all modules work correctly from an end-user's perspective.

Expected Outcome:

- BruhDive should work as a complete system.
- All pages should load without errors.
- Quizzes should function correctly, with the proper flow, question display, and scoring.
- User inputs (in forms and quizzes) should be validated, and all results should display correctly.
- Navigation and transitions between different parts of the app should be smooth.
- External integrations (e.g., API calls for fetching questions, EmailJS for feedback) should work as expected, including fallback mechanisms when APIs fail.

13.3. END-TO-END (E2E) TESTING

End-to-End (E2E) Testing focuses on testing the entire application from start to finish, simulating real user interactions with the system. The goal is to ensure that all components, both backend and frontend, work together seamlessly to deliver a smooth and functional user experience. For **BruhDive**, E2E testing involves testing the entire quiz process and user flows, ensuring that everything from quiz selection, answer submission, and result generation to interaction with the "Test Yourself" section works as intended.

Objective of E2E Testing for BruhDive:

The primary goal is to simulate how a real user would navigate through **BruhDive**, from accessing the platform to taking a quiz and receiving personalized career recommendations. It also involves validating the behavior of features like the "Test Yourself" API-powered quiz and ensuring that the data storage (via localStorage) works correctly.

Key Areas for E2E Testing in BruhDive:

1. **Quiz Selection and Navigation:**
   - **Start Quiz Flow:** Ensure that clicking on a language card (e.g., Python, JavaScript) correctly takes the user to the respective quiz page.
   - **Navigation Between Questions:** Ensure that users can smoothly navigate between quiz questions using the next and previous buttons.
   - **Submit Quiz:** Confirm that when the user completes the quiz, submitting it leads to the result page showing the score, level, strengths, and recommended topics.
2. **Answer Submission and Result Generation:**
   - **Answer Choices:** Test that selecting answers (correct and incorrect) updates the score and provides appropriate feedback to the user.
   - **Submit Answered Quiz:** After the user finishes the quiz, the system should calculate the score, display the user's level (e.g., Noob, Beginner, etc.), and show topic recommendations based on performance.
   - **Fallback Mechanism for API Failures:** In case of an API failure (e.g., for "Test Yourself"), the system should fallback to local storage questions and still generate meaningful results and recommendations.
3. **"Test Yourself" Section (Career Guidance):**
   - **Test Yourself API Flow:** Simulate the quiz taking experience in the “Test Yourself” section, ensuring that the AI-powered quiz correctly generates domain-based questions.

- **Career Guidance Accuracy:** Ensure that career recommendations (e.g., Web Developer, Data Scientist) are dynamically generated based on the user's highest-performing domain.
- **Fallback Questions Handling:** If the API fails, ensure that fallback questions are used and that career guidance is still generated based on those.

4. **Cross-Page Navigation:**
   - **Homepage to Quiz Navigation:** Ensure that all language cards on the homepage take the user to the correct quiz page.
   - **View Results:** Ensure that the user can view previous quiz results stored in localStorage, including scores, levels, and topic recommendations.
   - **Support and Resources Pages:** Validate that the support page (contact and feedback forms) and resources page (internships, learning resources, certifications) are accessible and function correctly.
5. **UI/UX and User Feedback:**
   - **Visual Feedback:** Ensure that feedback (e.g., score updates, confetti animations, success or failure messages) is shown clearly after quiz completion.
   - **UI Responsiveness:** Test the platform's responsiveness across various devices and screen sizes (especially mobile), ensuring the layout and content adjust properly.
   - **Smooth User Experience:** Ensure the user can easily navigate between pages and that the platform remains responsive during quiz-taking and result viewing.
6. **Local Storage Behavior:**
   - **Result Storage:** Ensure that quiz results are properly stored in localStorage so users can view their past quiz results even after refreshing the page.
   - **Test Data Persistence:** Verify that quiz data (score, level, strengths) persists in localStorage across different sessions.

TEST CASES:

Component Name: Main Screen

Purpose: To validate proper launch of the BruhDive application

Sr no.	**Test Case**	**Test Input**	**Expected Output**	**Actual Output**	**Remark**
1	To check if BruhDive is installed and opens correctly	Click on BruhDive icon twice	Dashboard of the application should be displayed	Dashboard is displayed clearly without fail	Passed
2	Check for the homepage 'Test Yourself' card	Click on 'Test Yourself' card	It should redirect to the quiz instructions screen	Redirected successfully to quiz screen	Passed

Component Name: Quiz Module

Purpose: To validate quiz launching and quiz question navigation

Sr no.	Test Case	Test Input	Expected Output	Actual Output	Remark
3	Check language quiz opens properly	Click on a language card (e.g., Python)	Quiz for selected language should start	Quiz started with questions displayed	Passed
4	Check if next question loads correctly	Click 'Next' after answering a question	Next question should be shown	Next question loaded properly	Passed

5	Check timer for quiz	Wait and observe during quiz	Timer should countdown correctly	Timer displayed and functional	Passed
6	Check if quiz ends after last question	Answer all 25 questions	Redirected to result page	Result page shown after last question	Passed
7	Check what happens if user refreshes in-between quiz	Refresh quiz page	Quiz restarts or continues from beginning	Quiz restarted as expected	Passed

Component Name: Result Module

Purpose: To validate performance analysis and result display

Sr no.	Test Case	Test Input	Expected Output	Actual Output	Remark
8	Check result shows after completing quiz	Finish all questions in a quiz	Score, level, and feedback should be shown	Result page with score and level displayed	Passed
9	Check if local storage stores results	Open browser dev tools > localStorage	Should show stored quiz data	Data stored successfully in localStorage	Passed

10	Check retake quiz option	Click on 'Retake Quiz' button	Quiz should restart from question 1	Quiz restarted as expected	Passed

Component Name: Test Yourself Module

Purpose: Validate AI-based quiz generation and career guidance

Sr no.	Test Case	Test Input	Expected Output	Actual Output	Remark
11	Check if questions load from API	Click 'Test Yourself' > Start Quiz	Questions from different domains should load	Questions loaded successfully	Passed
12	API fails - check fallback	Simulate API failure (offline mode)	Fallback questions should be shown	Fallback questions displayed	Passed
13	Check career guidance appears	Finish the AI quiz	Performance-based suggestions should be shown	Career advice and top domain displayed	Passed

Component Name: Resources Module

Purpose: Validate resource links and interaction

Sr no.	Test Case	Test Input	Expected Output	Actual Output	Remark

14	Check if Resources page opens	Click on 'Resources' in navbar	Page with learning resources should load	Resource list displayed correctly	Passed
15	Check if external links open	Click on an internship link	Should open in new tab	Link opened successfully in new tab	Passed

Component Name: Support Module

Purpose: Validate feedback and contact form functionality

Sr no.	Test Case	Test Input	Expected Output	Actual Output	Remark
16	Check if support page opens	Click on 'Support' in navbar	Contact & Feedback forms should be displayed	Support page opened with forms	Passed
17	Submit feedback form	Fill feedback and click submit	Feedback submitted (simulated)	Form reset after submit	Passed
18	Submit contact form	Fill details and click send	Should show thank-you alert or message	Message sent confirmation shown	Passed

13.4. MAINTENANCE

Maintenance refers to the process of updating, improving, or fixing a software application after its development is complete. Although BruhDive is not deployed on a live server or public platform, maintenance still plays a crucial role in ensuring its stability, scalability, and readiness for future development or deployment.

Types of Maintenance Applicable to BruhDive:

Type	**Description**	**Relevance to BruhDive**
Corrective Maintenance	Fixing bugs and errors found after development	Applicable – Bugs discovered during testing (UI glitches, logic issues, fallback handling) are addressed
Adaptive Maintenance	Modifying the system to adapt to changes in the environment (e.g., OS updates, browser versions)	Applicable – Ensuring compatibility with latest browsers, Node.js/React versions
Perfective Maintenance	Enhancing features or improving performance	Applicable – Improvements like UI enhancements, better fallback behavior, or adding new quiz domains
Preventive Maintenance	Making code changes to prevent future issues	Applicable – Code refactoring, improving comments, modularization for long-term maintainability

Maintenance Activities (Local Project Perspective):

- Refactoring code to improve readability and modularity
- Keeping React and Vite dependencies up to date
- Validating fallback question logic in offline mode
- Updating hardcoded data and resources (e.g., quiz topics, roadmaps)
- Improving performance and optimizing assets (images, CSS, etc.)
- Documenting code and functionality for easier handover or future use

Tools Used:

- **VS Code** for editing and local debugging
- **npm** for managing and updating packages
- **Chrome DevTools** for inspecting UI and performance issues
- **Git** for version control and backup

Note:

Since **BruhDive** is not deployed to a live production environment, **no server-side monitoring, user feedback loops, or downtime handling** are currently required. However, these may be included in future iterations upon deployment.

14. LIMITATION & ENHANCEMENT

➢ Limitations

- **Optimized for Web-Based Experience**: Currently, BruhDive is designed as a web-based platform. While this provides easy access across devices, future updates will explore the possibility of expanding to mobile applications for Android and iOS, offering users a more tailored experience on mobile devices.
- **Local Storage for User Data**: At present, user progress and quiz history are stored locally within the browser. This ensures ease of access, but in the future, we aim to integrate a cloud-based solution for saving user data across multiple devices, allowing users to access their quiz history seamlessly.
- **Focused Question Generation**: BruhDive's current quiz generation relies on a combination of hardcoded and API-generated questions. While this offers a solid foundation, plans are in place to incorporate dynamic question generation from a broader range of sources, ensuring a more diverse and adaptable quiz experience.
- **Career Guidance Based on Quiz Performance**: The current career guidance feature offers valuable insights based on quiz performance. However, we envision an enhanced, personalized guidance system in the future, which will factor in additional user attributes, such as career goals and interests, for more tailored career recommendations.
- **Dependence on External APIs**: BruhDive uses external APIs to generate questions and offer career advice. While this greatly enriches the platform, we plan to build more robust mechanisms for API backup and ensure continued functionality, even in the rare event of external service disruptions.

➢ Enhancement

To further improve the BruhDive experience, we have identified key areas for future growth:

- **User Authentication and Database Integration**: Future updates will introduce a secure login system, enabling users to create accounts, track progress, and save their quiz results. Integration with a database will allow for more personalized experiences and will help manage user data efficiently.

- **Dynamic and Expansive Question Generation**: To enhance the quiz-taking experience, BruhDive will incorporate a more dynamic backend system that generates fresh, real-time questions. This will allow the platform to continually evolve with new topics and domains, keeping quizzes relevant and engaging.
- **Advanced Career Guidance Engine**: In future versions, we plan to introduce a personalized career guidance system that considers various factors, such as individual preferences and long-term goals, alongside quiz performance. This will ensure that users receive highly relevant and actionable career advice.
- **Support for Additional Programming Languages and Domains**: BruhDive’s future releases will introduce quizzes for more programming languages and tech domains. This will enable users to explore a broader range of career paths, including emerging technologies such as data science, AI, and mobile app development.
- **Cross-Device Synchronization**: To enhance user convenience, future updates will include cross-device synchronization. This feature will allow users to continue their learning journey across multiple devices without losing progress, offering a seamless experience whether they’re on a desktop or mobile device.
- **Mobile Application Development**: BruhDive plans to extend its reach by developing mobile applications for Android and iOS. This will allow users to take quizzes and access resources on the go, providing even greater flexibility and engagement.
- **API Failover and Backup Solutions**: While the system currently relies on external APIs for dynamic content, we are working on building failover mechanisms to ensure continuous service. In the event of API disruptions, BruhDive will still provide users with quality content and functionality.

15. **BIBLIOGRAPHY**

- https://react.dev/learn
- http://www.google.com
- https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
- https://deepinfra.com
- https://www.emailjs.com/
- https://vite.dev/
- https://expressjs.com/