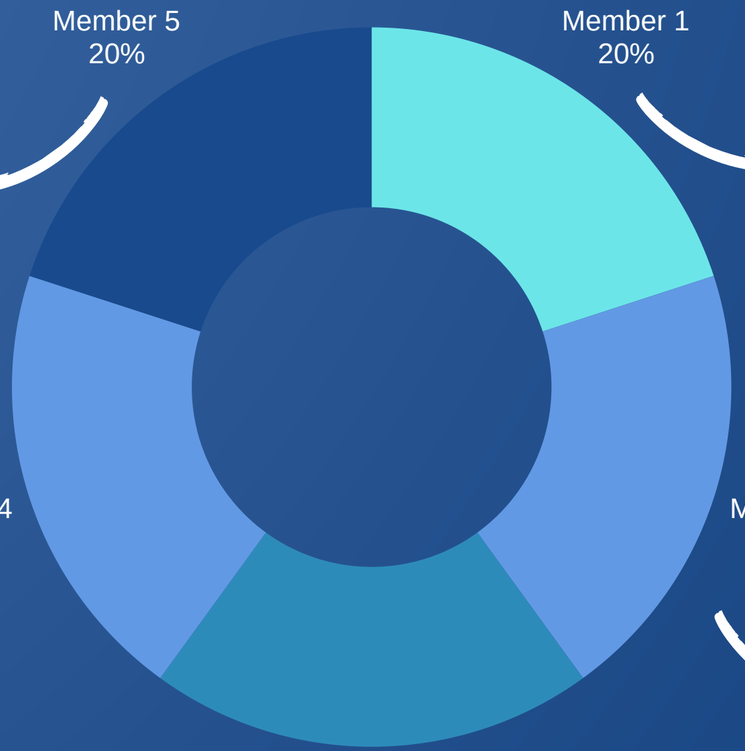
Formal Methods

MOBILE APPLICATION

GROUP MEMBERS

Hanzala Alvi
47583

Leena Sadiqa
46963



Jaweriya Khan
46549

Maryam Safdar
46481

Wafi Wahid
47322

TABLE OF CONTENT

FORMAL METHODS

"Formal methods are mathematical techniques used to specify, design, and verify software systems to ensure correctness, reliability, and security."

HOW FORMAL METHOD IS USED IN MOBILE APPLICATION

Leena Saddiqa

KEY CONCEPTS

- **Specification:**

"Formal methods start with precise, mathematical specifications of what the software should do."

- **Verification:**

"Techniques like theorem proving and model checking ensure the software meets its specifications."

- **Modeling:**

"We create models of the system’s behavior using tools like UML or TLA+."

- **Testing:**

"Formal methods help generate test cases to cover all possible scenarios."

- **Security and Safety:**

"They are used to verify security protocols and ensure compliance with safety standards."

EXPLANATION

- **Specification:**

The coffee machine should dispense coffee when the button is pressed.

- **Verification:**

Does the coffee machine really dispense coffee when the button is pressed.

- **Modeling:**

Drawing a diagram of the coffee machine's buttons and levers to see how they connect.

- **Testing:**

What if I press the button twice? Does the coffee machine still work correctly?

- **Security and Safety:**

What if someone tries to hack into the coffee machine? Can we prevent that?

WHY IS FORMAL METHODS IMPORTANT

Maryam Safdar

FORMAL METHOD IN MOBILE DEVELOPMENT

Formal methods (FM) use mathematical techniques to rigorously specify and verify system behavior. In mobile app development—where security, performance, and user trust are critical—FM can help ensure that the app's behavior matches its intended design, minimizes defects, and adheres to strict security requirements.

WHAT PROBLEMS CAN FORMAL METHODS (FM) HELP SOLVE?

Correctness and Reliability:

Clearly define app behavior to detect and fix bugs early, ensuring the app works as intended

Security and Safety:

Identify and prevent vulnerabilities before release, protecting against potential exploits.

Concurrency and Resource Management:

Ensure smooth operation of simultaneous processes without conflicts, preventing issues like deadlocks.

Detection of Algorithmic Bias:

Verify fairness in decision-making algorithms, promoting unbiased outcomes

RISKS OF NOT USING FORMAL METHODS (FM) IN MOBILE APP DEVELOPMENT

Hidden Bugs and Defects:
Without FM, subtle issues may go unnoticed, leading to app crashes or unexpected behavior.

Security Vulnerabilities:
Lack of rigorous analysis can leave exploitable gaps,risk of data breaches and unauthorized access

Algorithmic Bias:
Unchecked algorithms may produce unfair outcomes, damaging user trust and potentially violating regulations

Higher Maintenance Costs:
Post-release fixes are often more expensive; unreliable apps can harm reputation and incur significant remediation expenses.

ENSURING CORRECT AND UNBIASED OUTPUTS IN MOBILE APPS

Scenario:
A mobile healthcare app recommends personalized treatments using machine learning.

Specification:
Define precise guidelines

```mathematica
---------------- MODULE HealthcareApp ----------------

VARIABLE patientData, treatment

(* Specification: Treatment must follow clinical guidelines *)
SPECIFICATION SafeTreatment ==
    /\ treatment \in {"Treatment A", "Treatment B"}
    /\ patientData.symptoms /= NULL

(* Verification: Ensure Unbiased Output *)
ASSUME UnbiasedRecommendation ==
    /\ treatment = "Treatment A" => patientData.age > 18
    /\ treatment = "Treatment B" => patientData.age <= 18

(* Assurance: The system should never recommend an invalid treatment *)
THEOREM SafeTreatment => UnbiasedRecommendation

==================================
```

Verification:
Mathematically prove app's algorithms

Assurance:
Enhance user comply with regulations by minimizing errors

REAL-WORLD EXAMPLES & APPLICATIONS

Jaweriya Khan

IDENTIFICATION OF FORMAL METHODS

Formal Specification in Mobile Applications:
Formal specifications help define UI transitions, authentication mechanisms, and network interactions with clarity.

Formal Development in Mobile Applications:
Ensures that resource constraints (battery, network, CPU) are handled correctly while maintaining security and performance.

Formal Verification in Mobile Applications:
Formal verification ensures correctness in secure payments, cryptographic protocols, and safety-critical apps (e.g., medical or automotive applications).

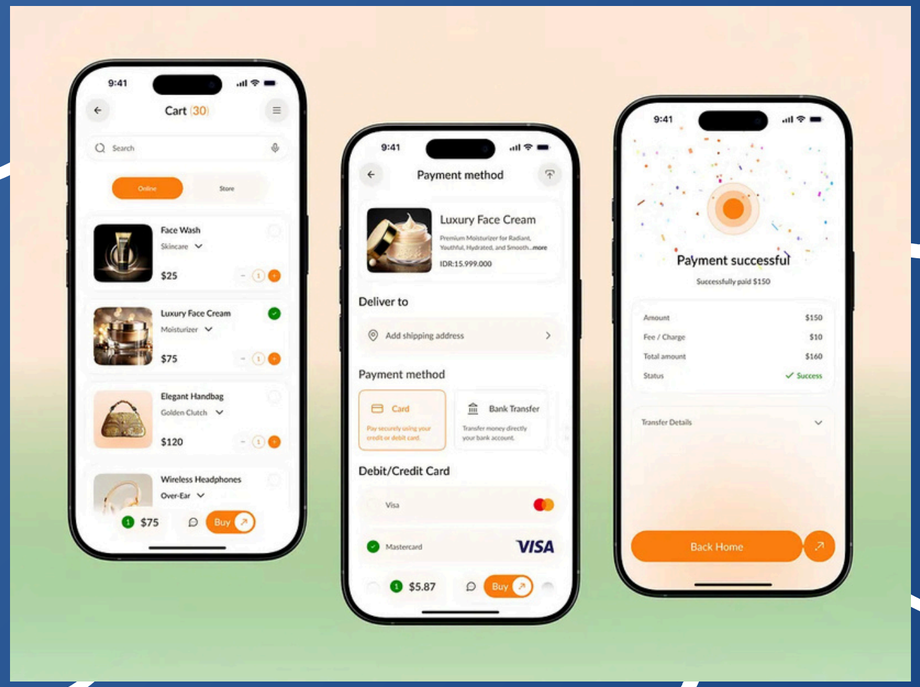
E-COMMERCE APP

Formal Method

Z Notation + VDM

Z Notation

Specify the state of the payment system (e.g., transactions, statuses like pending, successful, failed).



VDM

Model operations like "process payment" with preconditions (e.g., valid payment details) and postconditions (e.g., transaction status updated).

Verification

Prove that payment processing is secure and correct.

Outcomes

Ensures reliable and secure payment transactions.

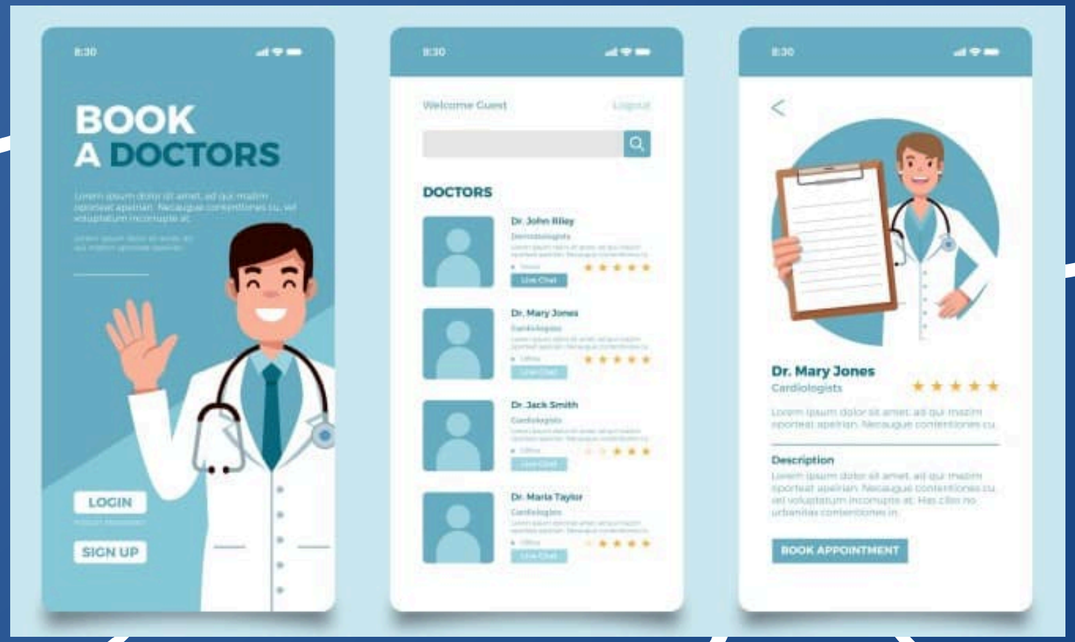
HEALTHCARE APPOINTMENT SCHEDULING APP

Formal Method

Z Notation + VDM + Algebraic Specifications

Z Notation

Define the system's state, including sets of patients, doctors, appointments, and schedules.



VDM

Model data structures like patient profiles, doctor availability, and appointment slots.

Verification

Prove that the scheduling algorithm behaves as intended (e.g., no overlapping appointments).

Algebric Specification

Specify operations like "book appointment" with preconditions (e.g., doctor is available) and postconditions (e.g., appointment added to the schedule).

RIDE-SHARING APPS

Formal Methods

Z Notation + VDM + Algebraic Specifications

Outcome

Ensures efficient and fair driver-passenger matching.

Z Notation

Specify the system's state (e.g., available drivers, passengers, locations).

VDM

Model operations like "match driver" with preconditions (e.g., valid locations) and postconditions (e.g., driver assigned to passenger).

Algebraic Specifications

Define the behavior of the matching algorithm (e.g., finding the closest driver).

PRACTICAL IMPLEMENTATION IN YOUR PROJECT

Wafi Wahid

I. Using FM in Mobile App Development

- **FORMAL METHODS (FM) ENSURE CORRECTNESS, RELIABILITY, AND SECURITY IN MOBILE APPLICATIONS.**

- Key areas of application:
  - Security: Define access control policies.
  - Concurrency: Model multi-threaded interactions.
  - Performance: Optimize resource consumption.
  - Error Handling: Detect edge cases systematically.

II. Choosing a Formal Method: Z Notation

- Why Z Notation?

- A mathematical specification language.
- Defines system properties rigorously before implementation.
- Helps with formal verification and refinement.

SECURE MOBILE BANKING APP

STEPS

- Define access control rules formally.
- Ensure only authorized users access sensitive operations.
- Model state transitions for authentication and transactions.
- Ensures only valid users can log in.
- Formalizes user authentication rules.

Z-NOTATION

Login Rule: If a user exists in the system, they can log in and will be added to the authenticated users list.

[USER]

AUTH_USERS: $\mathbb{Z} \rightarrow$ **USER**

set of authenticated users

AUTH_USERS is empty.

INIT AUTH_USERS = {}

Login $\exists$ **u: USER · (u** $\in$ **DOMAIN AUTH_USERS)** $\Rightarrow$ **AUTH_USERS' = AUTH_USERS** $\cup$ **{u}**

checks if a user exists in the domain

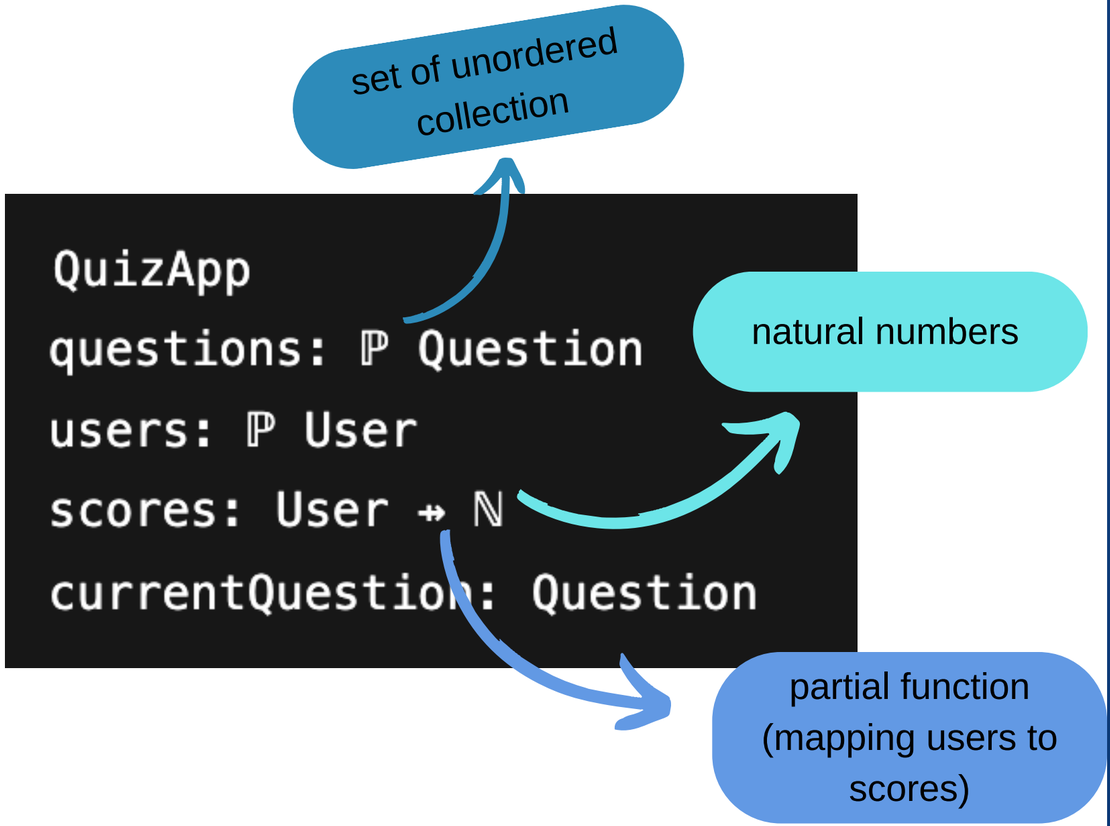
If the user exists, they are added

QUIZ APP IN REACT NATIVE

STATE

- **Questions:** A finite set of questions.
- **Users:** A set of registered users.
- **Scores:** A mapping of users to their scores.

Z-NOTATION

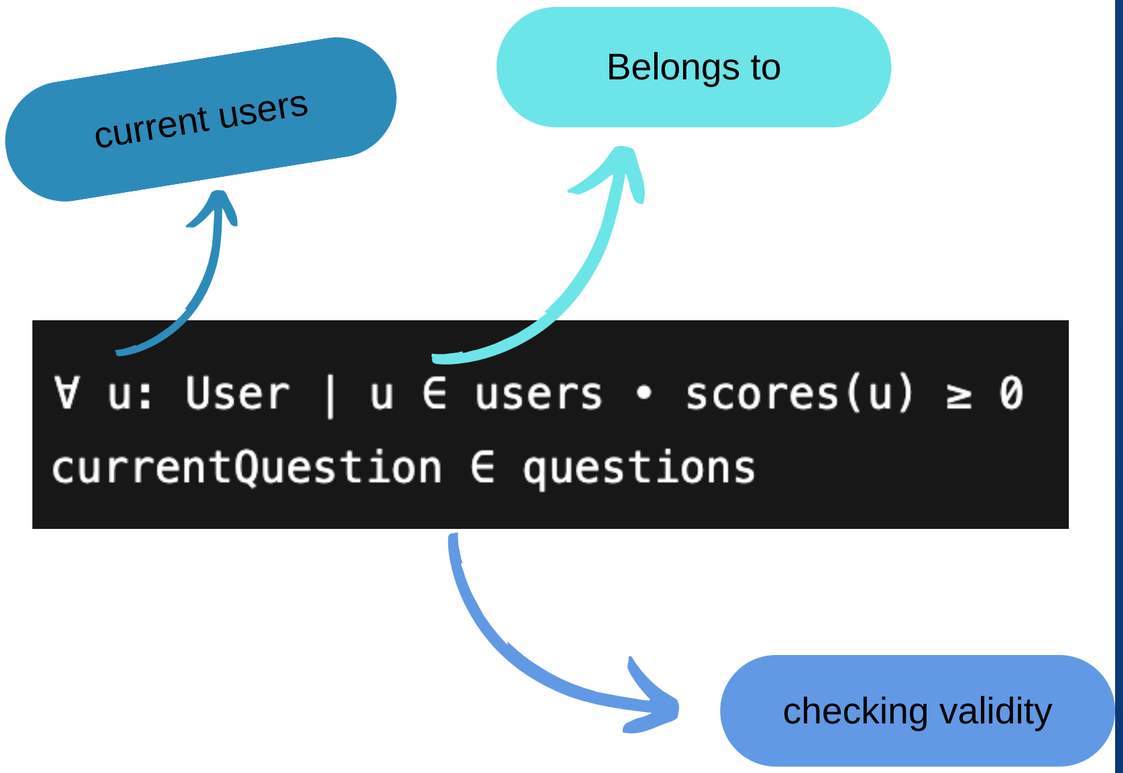


QUIZ APP IN REACT NATIVE

INVARIANTS

- A user’s score cannot be negative.
- The current question must exist in the question set.

Z-NOTATION



QUIZ APP IN REACT NATIVE

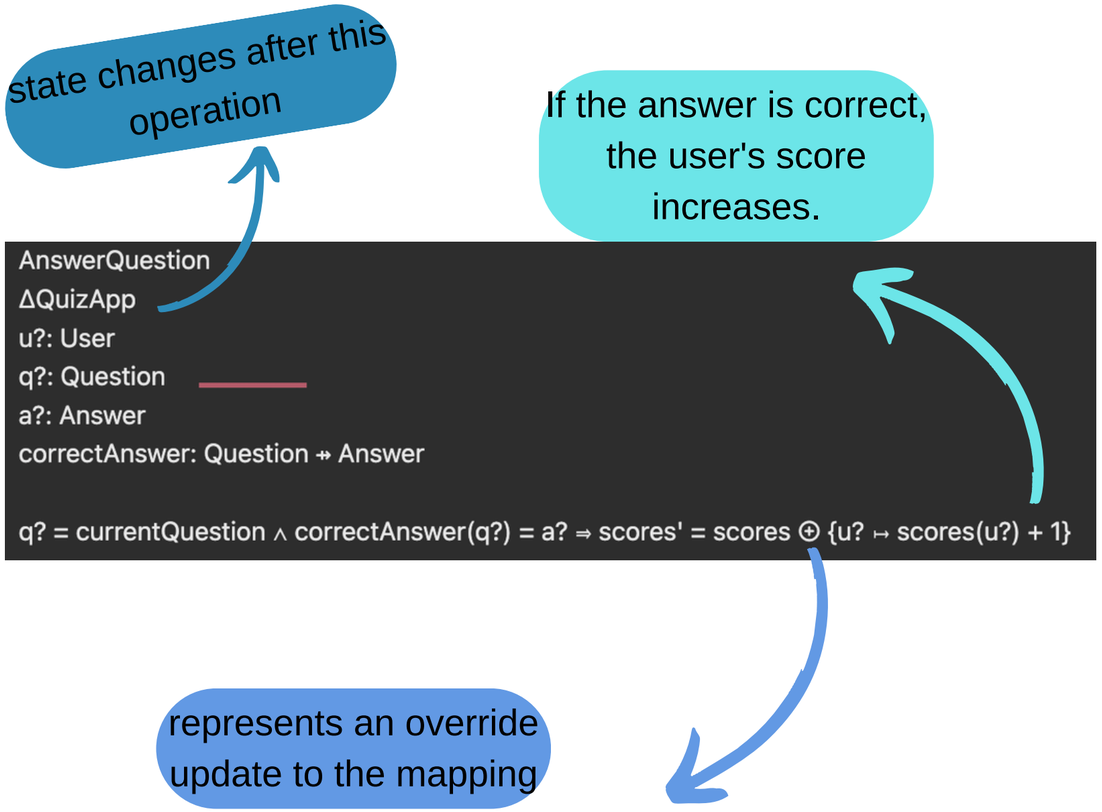
FUNCTIONAL SPECIFICATION

we define how the app behaves when users **answer a question, navigate, or exit.**

Answering a Question

- The system checks the answer.
- If correct, the user’s score is updated.

Z-NOTATION



QUIZ APP IN REACT NATIVE

STATES CHANGED

- Moving to the Next Question

Z-NOTATION

```
NextQuestion
ΔQuizApp
questions ≠ ∅ ⇒ currentQuestion' ≠ currentQuestion
```

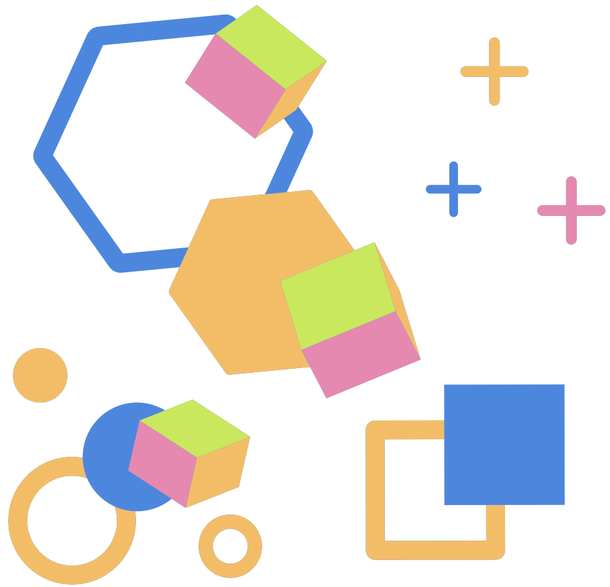
Ensures the next question is different.

QUIZ APP IN REACT NATIVE

VERIFICATION PHASE

- We use Z/Eves (a tool for Z-notation) to prove correctness:
- Check invariant preservation (e.g., score never goes negative).
- Validate state transitions (e.g., always selecting a valid question).

Z-NOTATION



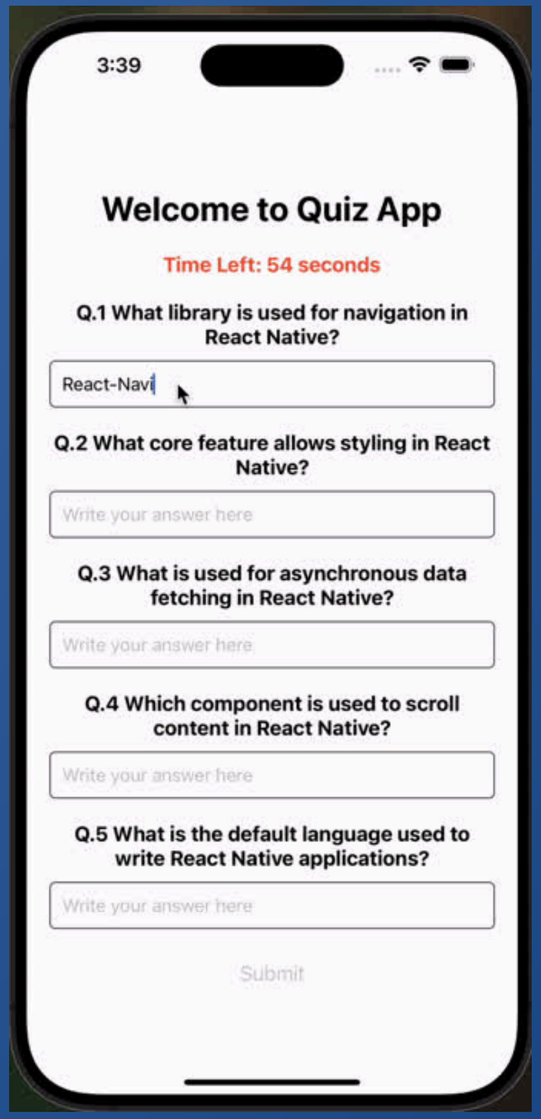
QUIZ APP IN REACT NATIVE

```
export default function App() {
  const [userAnswers, setUserAnswers] = useState(Array(quiz.length).fill(""));
  const [score, setScore] = useState(null);
  const [isSubmitted, setIsSubmitted] = useState(false);
  const [timeLeft, setTimeLeft] = useState(60);

  const handleInputChange = (text, index) => {
    const updatedAnswers = [...userAnswers];
    updatedAnswers[index] = text;
    setUserAnswers(updatedAnswers);
  };

  useEffect(() => {      You, 3 months ago • add timer
    if (timeLeft > 0 && !isSubmitted) {
      const timer = setTimeout(() => setTimeLeft(timeLeft - 1), 1000);
      return () => clearTimeout(timer);
    } else if (timeLeft === 0 && !isSubmitted) {
      handleSubmit();
    }
  }, [timeLeft, isSubmitted]);

  const handleSubmit = () => {
    let newScore = 0;
    quiz.forEach((q, index) => {
      if (userAnswers[index].trim().toLowerCase() === q.answer.toLowerCase()) {
        newScore++;
      }
    });
    setScore(newScore);
    setIsSubmitted(true);
    Alert.alert(
      "Quiz Completed",
      `Your final score is: ${newScore}/${quiz.length}`
    );
  };

  const isFormComplete = userAnswers.every((answer) => answer.trim() !== "");
```

QUIZ APP IN REACT NATIVE

```
return (
  <View style={globalStyles.container}>
    <Text style={globalStyles.welcomeText}>Welcome to Quiz App</Text>
    <Timer timeLeft={timeLeft} />
    {quiz.map((q, index) => (
      <QuizQuestion
        key={index}
        question={q}
        answer={q.answer}
        userAnswer={userAnswers[index]}
        onAnswerChange={handleInputChange}
        isSubmitted={isSubmitted}
      />
    ))}
    {!isSubmitted && (
      <Button
        onPress={handleSubmit}
        title="Submit"
        color="#841584"
        disabled={!isFormComplete || isSubmitted}
      />
    )}
    {score !== null && (
      <ScoreDisplay score={score} totalQuestions={quiz.length} />
    )}
  </View>
);
}
```

QUIZ APP IN REACT NATIVE



CHALLENGES AND LIMITATIONS

Hanzala Alvi

I. Why do many developers not use FM in their projects?

- Requires deep mathematical knowledge.
- Lack of awareness and industry adoption.
- Perceived as unnecessary for smaller projects.

II. BARRIERS TO ADOPTION

Complexity:
FM involves abstract mathematical models that are difficult to understand.

Time-Consuming:
Writing formal specifications takes significantly longer than traditional coding.

High Cost:
Requires specialized training and tools, making it expensive for small teams.

Tooling & Integration Issues:
Limited support in modern development environments.

Scalability Concerns:
Difficult to apply FM to large-scale, rapidly changing projects.

FUTURE OF FORMAL METHODS IN MOBILE APPLICATIONS

I. ENHANCING SOFTWARE DEVELOPMENT

- FM ensures:
  - Higher reliability: Reduces bugs and unexpected failures.
  - Better security: Provides mathematically proven correctness.
  - Scalability: Ensures large mobile systems remain maintainable.

- Adoption in:
  - Safety-critical apps (healthcare, finance, automotive).
  - AI-powered apps (ensuring ethical decision-making).
  - Blockchain and smart contracts in mobile applications.

II. ROLE OF AI & AUTOMATION IN FM

- **AI-powered Theorem Provers:** Automate FM verification.
- **Machine Learning + FM:** Predict system failures and improve verification speed.
- **Automated Code Generation:** Convert formal specifications into executable code.
- **Future Trend:** Seamless integration of FM into agile development workflows.

REFERENCES

- Lamport, L. (2002). Specifying systems: The TLA+ language and tools for hardware and software engineers. Addison-Wesley.
- Bowen, J. P., & Hinchey, M. G. (1995). Formal methods in safety-critical systems. Springer.
- Hoare, C. A. R. (1985). Communicating sequential processes. Prentice Hall.
- ACM Digital Library. (2023). Formal methods in mobile development. Retrieved fromhttps://dl.acm.org/doi/10.1145/3579856.3596440
- ResearchGate. (2018). A systematic literature review of the use of formal methods in medical software systems. Retrieved from https://www.researchgate.net/publication/323427779

GITHUB LINK

- https://github.com/Wafi-wahid/QuizApp-ReactNative

THANK YOU