**Overview:**  
Testing is a very important part of software development which ensures that each component of the app performs correctly, reliably, as expected. It provides early detection of bugs, helps validate business logic, ensures stability under edge conditions and cases and contributes to a better user experience. For mobile applications with server-side interactions and third-party integrations, testing becomes even more critical to catch integration issues, data consistency problems and ensure responsiveness across devices and network conditions.

**Overview of the Application’s Functionalities**

The application developed for this Bachelor’s thesis is a cross-platform mobile app built using Flutter for the frontend and FastAPI with MongoDB for the backend. The main functionalities of the app include:

-User Authentication: Secured login and registration with authentication based on tokens

-Financial Quiz Section: 10 multiple-choice financial literacy questions are generated using the Gemini LLM API 2.0 Flash. If the API is unavailable or overloaded, a set of fallback static questions is used. Users receive a virtual amount of money as reward for answering at least 9 questions correctly, the amount differs based on the selected difficulty level

-Demo Investing Platform: Users can search for stock symbols. Stock data such as current price and historical performance is retrieved using the yfinance library. Users can buy or sell stocks using virtual currency. Their portfolios are stored and updated in the MongoDB database, they contain the companies tickers together with the quantities owned by the user

-AI-Based Stock Prediction: Uses a custom LSTM-based model to predict stock trends (10-day or 2-year horizons). In addition, sentiment analysis is applied to news articles using a language model, assigning scores for sentiment (-1 to 1) and credibility (0 to 1). These scores influence the predicted prices.

**Testing Techniques**

A variety of testing strategies should be applied during the development process to validate both individual features and the system as a whole:

1. **Unit Testing**: Unit testing focuses on the smallest testable parts of the application. In this project, it will validate:

-Backend API endpoints (authentication logic – login and register, quiz generation including the fallback logic, prediction adjustments)

-LSTM stock price predictor (ensuring output format and shape correctness)

-Sentiment and credibility analysis: check that the range of the score is as expected and it is following the constraints

-Quiz reward evaluation logic (correct scoring and reward assignment)

-Flutter widgets such as form validators, button state changes or state provider updates.

**2. Integration Testing**: Integration testing ensures that multiple components work together correctly. It was used to test:

-Frontend - Backend communication (ex: quiz result submission, stock purchase requests)

-Database interaction: Ensuring MongoDB updates correctly reflect user actions

-Authentication flow: Validating token usage across screens and logout handling.

**3. Testing the Widgets / UI (Flutter)**: Flutter provides a robust testing framework for verifying widget behavior. UI tests include:

-Checking loading indicators during network calls

-Testing dynamic button enabling / disabling based on form state

-Ensuring the portfolio view reflects data updates after buy/sell actions (update quantity, average buy price).

**4. System Testing:** System testing validates the end-to-end functionality of the entire application.Test Scenarios:

-Correct Path: A user signs up, takes a quiz, earns virtual currency, invests in stocks, and views predictions

-Fallback Handling: The quiz system correctly switches to fallback mode when Gemini fails and retrieves the quiz consisting of the 10 hardocded questions

-Edge Cases: Testing behavior with no internet connection, expired JWT tokens, or invalid ticker input.

**5. Exploratory Testing:** Exploratory testing is performed to simulate real-world behavior and discover edge cases not covered by automated tests. This include:

-Rapid navigation between screens

-Inputting malformed data (ex: empty quiz answers, invalid ticker formats)

-Interrupting requests mid-operation (e.g., switching screens during a fetch).

**6. Performance Testing:** While not a core part of the thesis, performance testing is considered for:

-API latency under concurrent quiz submissions

-Stock price data retrieval time from yfinance, Alpha Vantage and FMP APIs