1. Briefly explain each level of testing, using the testing pyramid as a reference.

Unit Testing:

- Forms the base of the testing pyramid;
- Has a limited scope;
- Evaluates a single variable or component;
- Doesn’t rely on external dependencies;
- Tests are fast and easy to write;
- Typically written by developers;
- The use of simulators and emulators is recommended.

Integration Testing:

- Second layer in the testing pyramid;
- Verifies interactions between different parts of the application;
- Tests communication with external components;
- Ensures the software communicates effectively and receives/provides accurate data;
- Slower than unit tests.

End-to-End Testing:

- Covers most or all of the application code;
- Executed in a staging environment;
- Expensive to maintain;
- Slower to run;
- Represents the top layer of the testing pyramid.

2. Let's suppose we have a banking app under development and the business decides to implement a search bar. A development task is created, and you are assigned to test this functionality. Describe the full process from start to finish, including writing test cases, actual testing, closing the task, and what is required for the task to be considered complete.

Understanding the requirements:

Functional:

- Search using keywords;
- Verify the search bar is visible on the page;
- Check if characters can be typed into the search bar;
- Validate the accuracy of displayed results;
- Verify advanced search options (deposits, withdrawals);
- Check what happens when nothing or special characters are entered.

Non-functional:

- Check response time;
- Assess usability;
- Test on multiple devices.

Discussions with developers regarding unclear functionalities:

- Search criteria;
- Types of searches;
- Placement in the app.

Writing the test cases:

Unit Testing: test functions or pieces of code:

- Validate that functions return correct values;
- Confirm that results are displayed;
- Ensure filtering works properly.

Integration Testing: check how the search function integrates with other components:

- Database interactions;
- Format and presentation of results.

End-to-End Testing: test the entire flow:

- Open app → type in search bar → click button → observe results;
- Test behavior when not logged in;
- Validate what happens when there are no results.

Executing the tests:

- Start actual testing based on the test scenarios.

Reporting bugs:

- Document each bug;
- Include screenshots if necessary.

Retesting after fixes:

- Re-execute tests to confirm bugs are fixed;
- Update the status of test cases accordingly.

Validating results:

- Confirm with stakeholders that the functionality meets requirements.

Closing the task:

- Once validated, mark the task as complete.

Conditions for closing the task:

- Function meets all requirements;
- Approved by stakeholders;
- Is functional;
- Test cases are finalized and closed.

3. What is the advantage of unit testing compared to end-to-end/system testing?

Unit Testing – Advantages:

- Faster execution, written early in the development phase, allows early bug detection;
- Lower cost due to simpler implementation and fewer resources;
- More efficient since they focus on isolated pieces of code;
- Larger number of tests;
- Improves code quality.

4. What is the advantage of end-to-end/system testing compared to unit testing?

End-to-End/System Testing – Advantages:

- Simulates real user behavior;
- Performed in staging environments;

- Identifies integration issues;
- Offers a broader overview of the application;
- Helps validate requirements.

5. You have the following functionalities in the application:

- Loan calculation;
- Incident reporting (errors related to payments, functionality, display);
- All fields in the registration page are mandatory;
- IBAN payments;
- Investments via the banking app.

Assign appropriate testing levels (unit, integration, end-to-end) to each functionality and justify your choices.

Loan calculation:

- *Unit Testing:* Validate the calculation logic at code level.
- *Integration Testing:* Check how the function integrates with other components.
- *End-to-End Testing:* Simulate real user loan calculation scenarios in the app.

Incident reporting:

- *Unit Testing:* Test the core logic behind incident submission.
- *Integration Testing:* Verify how the reporting system interacts with error-handling modules.
- *End-to-End Testing:* Ensure the full incident report process works and is user-friendly.

Mandatory registration fields:

- *Unit Testing:* Validate each field individually and the logic behind validation.

- *Integration Testing:* Confirm correct integration of the validation system.
- *End-to-End Testing:* Test the full flow for completing the form and receiving feedback.

IBAN payments:

- *Unit Testing:* Validate the logic and correctness of IBAN processing.
- *Integration Testing:* Test interaction with databases and transaction systems.
- *End-to-End Testing:* Check the full payment process from input to confirmation.

Investments via the banking app:

- *Unit Testing:* Test the investment calculation logic.
- *Integration Testing:* Verify correct integration with the investment platform.
- *End-to-End Testing:* Simulate the full investment process and check overall functionality.