MISSION	STRATEGY	SCOPE	PROBLEMS
Business Value This testing approach will assist to: - locate defects early, - guarantee reliability of the system - ensure a smooth operation. Providing a high-quality system will surely improve user satisfaction and engagement, especially in the case of education workflows, thus the educational environment will be better. This means that the product is stable even on different platforms, thus allowing the project to be delivered on schedule.	**Approach** 1. Agile Testing Framework 2. Automated Testing 3. Manual testing 4. Performance and load testing 5. Cross-Platform Testing 6. Security Testing 7. Documentation and reporting	**In Scope** The in scope section covers the features, modules, and testing activities that are essential to the success of the project. These are in line with the basic system functions and user needs for Teacher Edition (in 6 months) and Student Edition (in 12 months).	**Risks** Early identification and mitigation of risks is essential to ensure system success. 1. Communication failure between services 2. Performance under Load 3. Inconsistencies between platforms 4. Security bugs 5. Complexity of Scheduling and Alerting Features
Testing Goals The testing goals are designed to align with the project's mission, focusing on delivering a robust, reliable, and high-performance system that meets the needs of different user groups. The goals are structured to ensure comprehensive coverage and fast feedback loops, critical for Agile	**Metrics** To ensure that testing is effective, measurable and continuously improving, the following metrics will be monitored and communicated to stakeholders: 1. Test coverage 2. Defect density 3. Mean Time to Detection (MTTD) and Mean Time to	**Out of scope** The out of scope section defines areas that will not be included in the current testing to save time and resources. These may include features that are not a priority for the current release, or aspects of the system that will remain untouched for the time being.	**Blind Spots** Blind spots are areas where more clarity is needed or where potential risks may not yet be fully visible. These are important to consider when planning the testing strategy, as they represent unknowns that could affect the project later. 1. Integration with external systems

development environments. 1. Comprehensive Coverage of Core Functionalities 2. Common Platform Compatibility 3. Early Detection of Defects 4. Stable and Scalable Performance 5. Automated Regression Covering 6. Security and Data Privacy	Resolution (MTTR) 4. Success/failure rate of automated test suites 5. Performance metrics 6. Course of test execution 7. Error reopening rate **Entry / Exit Criteries** **Entry Criteria.** Testing for different phases or feature sets can only begin once certain prerequisites are met. The entry criteria ensure that the environment is ready and that the team can begin testing without roadblocks: 1. For System Testing 2. For Performance Testing 3. For Regression Testing **Exit Criteria.** Testing can only be considered complete when specific conditions are met to ensure the system's readiness for release. These criteria provide a clear stopping point for testing: 1. For System Testing		2. Mobile Platform Specific Behavior 3. Load Patterns 4. User feedback on usability

	2. For Performance Testing 3. For Release (Teachers & Students)		

MISSION

Business Value

This testing approach will assist to:
- locate defects early,
- guarantee reliability of the system
- ensure a smooth operation.

Providing a high-quality system will surely improve user satisfaction and engagement, especially in the case of education workflows, thus the educational environment will be better. This means that the product is stable even on different platforms, thus allowing the project to be delivered on schedule.

Testing Goals

The testing goals are designed to align with the project's mission, focusing on delivering a robust, reliable, and high-performance system that meets the needs of different user groups. The goals are structured to ensure comprehensive coverage and fast feedback loops, critical for Agile development environments.

1. **Comprehensive Coverage of Core Functionalities**

Ensure that scheduling, task submission, grading, feedback, and notifications critical features are covered through robust testing across all platforms.

- Objective: If we have a minimum of 90% test coverage of the area, then the components will be

guaranteed to work properly (for instance, students submitting tasks, teachers grading, and admins scheduling events).

- How: Along with the use of manual and automated functional testing, both expected and edge case scenarios will be considered in this case.

2. Common Platform Compatibility

Ensure a smooth and consistent experience for users across all platforms including web, iOS, Android and Windows app.

- Objective: Check that main user actions and user interface components work the same on all devices and browsers. Platform-related bugs will also be recognized and solved.
- How: Exact cross-platform testing will be used to guarantee the same quality of appearance, performance, and functionality across the platforms using tools for mobile device testing and web browser compatibility checking.

3. Early Detection of Defects

It is critical to find and fix the defects as early as possible in the development lifecycle in order to reduce the cost of fixing issues later in the project.

- Objective: Conduct thorough unit, integration, and API testing for each sprint to reduce the number of defects found in later stages of testing (e.g. regression testing, user acceptance testing).
- How: The testing team will catch the defects as soon as the new features are developed by using continuous testing within sprints, automated regression suites, and early integration testing.

4. Stable and Scalable Performance

It should be guaranteed that the system runs perfectly under different load conditions, especially during the peak usage times (e.g. the assignment deadlines or when the multiple notifications are triggered).

- Objective: The system should meet the performance benchmarks, such as response times less than 2 seconds for the key actions (task submission, schedule loading) and be stable under stress testing (e.g. simulating thousands of concurrent users).

- How: Performance testing will be done by using load testing tools (e.g. JMeter, Locust) and by simulating real-world scenarios to ensure that the backend can scale effectively using Kubernetes for auto-scaling.

5. Automated Regression Covering

Be prepared and keep a thorough suite of automated regression tests to guarantee that new development will not cause bugs to the already existing functionality.

- Objective: Develop automated test scripts that cover vital workflows and ensure that every sprint finishes with a passing suite of regression tests.
- How: The regression suite will be developed progressively, as new features are added, and it will be run automatically in the CI/CD pipeline after every build.

6. Security and Data Privacy

Make sure that the system meets data security best practices, including the protection of sensitive student and teacher data, and the implementation of strong role-based access control.

- Objective: Make sure that all data is encrypted during transmission (e.g., using HTTPS) and that role-based access control functions as intended to prevent unauthorized access.
- How: The security testing will mainly check that only authorized users (students, teachers, and admins) can perform certain actions, thus ensuring that the data is protected from unauthorized access or tampering.

STRATEGY

Approach

1. Agile Testing Framework

The Agile development process will be testing incorporated, that way the smooth collaboration continues between testing, development, and product teams. Each sprint will include planning and execution of testing activities, thus the functionality of important features is confirmed before moving to the next sprint.

- Scrum meetings will be organized on a daily basis to allow for the test team to update on the ongoing testing processes as well as to discuss the test obstacles.
- Sprint Planning & Retrospectives will be done by the test team. This will help in having the testing needs considered in the beginning which will help them in implementing improvements later on.

2. Automated Testing

A large part of regression, API, and performance testing will be automated to ensure that the testing is efficient, repeatable, and scalable.

- API Testing: The automated tests of the Spring Java backend and React TS frontend will go through all major interactions between the microservices in the systems to trace and correct communication as well as to ensure that the units are stable.

3. Manual testing

Certain areas, especially UI/UX validation and exploratory testing, will be done manually. These tests are necessary to ensure that the system is user-friendly, intuitive and meets users' expectations.

• Exploratory testing: Used to explore edge cases and less structured user flows to catch issues that automated tests

might miss.
• Usability testing: Manual tests evaluate the user experience and ensure that the interface is intuitive for students, teachers and administrators.

4. Performance and load testing

Performance testing will simulate peak load conditions, especially during high-traffic scenarios such as assigning assignments or school-wide announcements. Stress tests ensure that the system remains responsive and stable.
• Tools like JMeter and Locust will be used to simulate high traffic, especially during critical events (eg task deadlines, display schedules).

5. Cross-Platform Testing

• Web, iOS, Android: Dedicated tests ensure feature parity and consistent user experience across all platforms using device simulators and real-device testing for mobile platforms.
• Windows Delphi App: Even if no new features are developed, integration testing will ensure compatibility with new backend services.

6. Security Testing

Focus on role-based access control authentication (students, teachers, administrators) and ensuring data privacy, especially for sensitive student records. Basic security testing will include:
• Authentication and authorization checks ensure that role-based permissions work as expected.
• Encryption Verification to ensure encryption of all data transfers (especially between microservices and frontend) using HTTPS.

7. Documentation and reporting

All test activities, defects and test results will be documented using the test management tool. This ensures traceability and transparency with stakeholders.
• Test cases will be tracked in a central system such as JIRA along with development tasks.
• Bug reporting: Bugs will be categorized by severity and impact and communicated to the development team for quick resolution.

Metrics

To ensure that testing is effective, measurable and continuously improving, the following metrics will be monitored and communicated to stakeholders:

1. Test coverage

Percentage of critical system functions covered by tests (automated and manual).

• Goal: Achieve at least 90% coverage of basic functions (assignment submission, assessment, planning, notifications).

2. Defect density

The number of defects found per module or feature, which helps identify high-risk areas.

• Goal: Keep defect density below 2 critical defects per element.

3. Mean Time to Detection (MTTD) and Mean Time to Resolution (MTTR)

These metrics track how quickly defects are identified and resolved. A lower MTTD/MTTR indicates a more sensitive test process.

• Goal: Maintain mean time to detection < 1 sprint cycle and mean time to resolve < 2 sprint cycles for critical issues.

4. Success/failure rate of automated test suites:

Track the success of automated test suites (eg API, regression).

• Goal: Maintain more than 95% throughput on critical routes.

5. Performance metrics

Measurement of system response under load (e.g. job submission times, API response times).

• Goal: Maintain response times below 2 seconds for key workflows during peak workloads.

6. Course of test execution

Track the number of test cases executed vs. planned during each sprint.

• Goal: Ensure 100% execution of planned test cases by the end of each sprint.

7. Error reopening rate

Track how often resolved bugs are reopened, indicating ineffective fixes.

• Goal: Keep the bug reopen rate below 10%

Entry / Exit Criteries

Entry Criteria

Testing for different phases or feature sets can only begin once certain prerequisites are met. The entry criteria ensure that the environment is ready and that the team can begin testing without roadblocks:

1. **For System Testing:**

   - Codebase is successfully built and deployed in the test environment.
   - All unit tests have passed.
   - APIs and backend services are integrated and functional in the test environment.
   - Test cases are written and ready for execution.

2. **For Performance Testing:**

   - Basic functional testing has been completed, and the system is stable.
   - Test environment is configured to simulate peak loads (e.g., simulated user accounts, preloaded data).

3. **For Regression Testing:**

   - The latest features are implemented, and a stable build is available.
   - Automated test scripts are updated to include all relevant features for regression.
   - Integration tests between microservices and frontend components have passed.

Exit Criteria

Testing can only be considered complete when specific conditions are met to ensure the system's readiness for release. These criteria provide a clear stopping point for testing:

1. **For System Testing:**

   - All critical and high-priority test cases have passed.
   - No high or critical severity defects are open.
   - Test coverage is at least 90% for core functionalities.
   - Performance benchmarks (response times, load capacity) are met.
   - Regression test suites pass with no significant issues.

2. **For Performance Testing:**

   - The system performs within acceptable limits under peak load conditions.
   - No performance bottlenecks or crashes occur during stress testing.

3. **For Release (Teachers & Students):**

   - All identified defects are either resolved or accepted with mitigation plans in place.
   - UAT (User Acceptance Testing) is completed, and stakeholders approve the release.
   - Regression tests pass without critical failures.
   - No blocking issues are open, and all non-critical defects have been documented for future sprints.

SCOPE

In Scope

The in scope section covers the features, modules, and testing activities that are essential to the success of the project. These are in line with the basic system functions and user needs for Teacher Edition (in 6 months) and Student Edition (in 12 months).

1. **Edition for teachers (in 6 months)**

- Task Management:

Teachers will be able to create, assign and grade assignments and homework for students. Testing will focus on ensuring that tasks with specific deadlines can be assigned and evaluated quickly.

- Rating and Feedback:

Teachers should be able to evaluate student contributions and provide detailed feedback. This includes testing different rating scales, comments and feedback mechanisms.

- Schedule Planning:

Teachers and administrators can create, update and manage lesson plans and special events. Testing will include dynamic scheduling, conflict resolution, and calendar management.

-**Notice:**

Teachers and administrators should receive notifications when key actions occur (eg assignments are submitted, grades are updated). Testing will focus on the delivery and timing of notifications across all platforms.

- Cross Platform Features:

All of the above functionality must work consistently across web apps (React TS), mobile apps (iOS and Android), and the legacy Delphi app for Windows. Testing will ensure consistent user experience and functionality across all platforms.

- **Backend Functions (Spring Java)**

Testing will ensure smooth communication between the backend microservices and ensure that the APIs work correctly for processing student submissions, teacher evaluations, and schedule updates.

- **Database Integrity (PostgreSQL):**

Tests verify that data (eg, student records, grades, assignments) is correctly stored, retrieved, and managed across all user interactions.

- **Performance Testing:**

Load and stress testing to ensure the system can handle a large number of concurrent users (eg at peak times such as assignment deadlines or school-wide events).

- **Security Testing:**

Role-based access control will be tested to ensure that only authorized users can access or edit certain features (eg teachers managing assessments, administrators editing timetables).

2. **Student edition (in 12 months)**

- **Homework:**

Students will be able to submit homework through integrated editors. Testing will verify different submission formats (text, files, multimedia) and their compatibility across platforms.

- **Viewing Schedules:**

Students should have access to class schedules and event announcements. Testing will focus on the correct display of schedules and any changes made by teachers or administrators.

- **Notice:**

Students should receive notifications about grading updates, upcoming lessons, homework due dates, and feedback. Testing notifications across platforms ensures consistency across devices.

- **Legacy Delphi applications for Windows:**

The application will continue to be maintained and compatibility with new backend services (Spring Java) will

be ensured. Testing will focus on integration and stability, although no new features will be developed.

- **Cross-Platform Testing (Web, iOS, Android, Windows):**

  Testing will ensure that features for students are accessible and consistent across all platforms. This includes UI/UX testing to ensure a seamless experience for students on a variety of devices.

- **API Testing (REST):**

  Ensure APIs handling student data, submissions, and notifications work as expected and maintain data integrity across services.

- **Performance Testing:**

  As with teacher layoffs, performance tests ensure that the system can accommodate a large number of simultaneous users, especially during exam periods or deadlines.

Out of scope

The out of scope section defines areas that will not be included in the current testing to save time and resources. These may include features that are not a priority for the current release, or aspects of the system that will remain untouched for the time being.

1. **Development of new features for Delphi application for Windows:**
   - No new features will be developed for the older Windows Delphi application. Testing will only focus on maintaining the current feature set and ensuring compatibility with new backend services.
2. **Penetration and Advanced Security Testing:**
   - While basic security testing (e.g. role-based access control, encryption) is in scope, advanced security testing

such as penetration testing and in-depth vulnerability assessment are excluded for the first releases.

3. **Comprehensive data migration testing:**
   - The scope does not include testing of any extensive data migration activities from legacy systems or databases, if such migration occurs. Emphasis will be placed on current, ongoing data integrity.

4. **Non-critical mobile features (iOS/Android):**
   - Non-critical features such as advanced mobile-specific gestures, push notifications with multimedia, and offline capabilities for mobile apps will not be tested in early releases. Initially, only the basic functions will be covered.

5. **Integration of third-party tools:**
   - Any integration with third-party tools (eg integration of external calendars or file storage services such as Google Drive or Dropbox) will not be included in this version. If implemented, they will be tested in future iterations.

6. **Localization and Multilingual Testing:**
   - Testing for multiple languages, internationalization and region-specific settings (eg date formats, language preferences) is out of scope at this stage and will be considered in future releases.

PROBLEMS

Risks

Early identification and mitigation of risks is essential to ensure system success.

1. Communication failure between services

- Risk: The microservices architecture of the system relies heavily on REST APIs for communication between components such as backend (Spring Java) and frontend (React TS) as well as mobile applications. Any API call failure can disrupt critical workflows such as task submissions or schedule updates.
- Impact: If the service fails, critical functionality (eg students submitting homework or teachers updating grades) may be compromised, leading to user frustration and delayed processes.
- Mitigation: Implement automated API testing with comprehensive coverage of all services. Use service-level monitoring (eg Kibana) to detect failures early. Provide robust error handling and retry critical API calls.

2. Performance under Load

- Risk: The system is expected to handle a large number of simultaneous users, especially during peak times (eg assignment deadlines, school events). The backend may struggle to scale efficiently during high-traffic scenarios, resulting in reduced performance.
- Impact: Users may experience slow response, crashes, or delays in receiving notifications, which may severely impact the user experience.
- Mitigation: Perform load and stress tests to identify performance bottlenecks. Use Kubernetes to dynamically

scale services during peak loads. Optimize the use of RabbitMQ for managing queues with large message volumes.

3. Inconsistencies between platforms

- Risk: There is a risk that the user experience may differ across platforms (web, iOS, Android, legacy Windows apps), resulting in inconsistent behavior, layout, or functionality.
- Impact: Cross-platform inconsistencies could frustrate users when switching between devices, leading to a lack of confidence in the reliability and usability of the system.
- Mitigation: Implement thorough testing across platforms to ensure consistency of user experience. Use automated tools to test across multiple devices and browsers and resolve UI/UX inconsistencies early in development.

4. Security bugs

- Risk: Handling sensitive data of students and teachers represents a significant security risk. Without adequate security measures, a breach of personal data or unauthorized access to critical functions (e.g. evaluation, planning) could occur.
- Impact: A data breach can lead to loss of trust, non-compliance and damage to reputation.
- Mitigation: Implement basic security testing focusing on encryption, role-based access control and authentication. Regularly review and update security policies and ensure HTTPS is enforced in all communication channels.

5.Complexity of Scheduling and Alerting Features

- Risk: Scheduling and notification systems are critical but complex to implement, especially when managing different time zones, concurrent events, and recurring tasks. Errors in planning or lack of notice could significantly disrupt school operations.
- Impact: Incorrect schedules or missing notifications could lead to confusion among users (teachers, students,

administrators), which would undermine trust in the system.
- Mitigation: Extensive functional testing of scheduling features with various edge cases (conflicting schedules, holidays, time zone differences). Implement real-time notification delivery tracking to ensure fast resolution of issues.

Blind Spots

Blind spots are areas where more clarity is needed or where potential risks may not yet be fully visible. These are important to consider when planning the testing strategy, as they represent unknowns that could affect the project later.

1. Integration with external systems
- Blind Spot: The project scope does not mention third-party integrations (eg with external file storage services, authentication systems or calendar applications), but if such integrations are added later, they may cause unexpected problems.
- Impact: External dependencies may cause delays or crashes, especially if their APIs are unstable or have conflicting configurations with the system.
- Solution: Get clarity from stakeholders on whether external integrations are expected, and if so, perform early exploratory testing of those integrations to identify potential issues.

2. Mobile Platform Specific Behavior

- Blind Spot: Certain platform-specific behaviors (eg iOS vs Android) may not have been fully accounted for during development. Differences in notification handling, background processes, and user permissions can cause inconsistencies.
- Impact: Problems can occur if the behavior of mobile apps differs on different platforms, leading to platform-specific bugs that can be difficult to reproduce and fix.
- Solution: Investigate early in the development phase and clarify any platform-specific requirements or limitations. Use mobile-specific test frameworks to ensure consistent behavior across iOS and Android.

3. Load Patterns

- Blind Spot: It is not clear what the expected load will be during peak times as the actual usage patterns are not fully understood at this stage. Peaks may be higher than expected, especially around exam or assignment deadlines.
- Impact: Unexpected load fluctuations can cause performance issues as the system may not be properly scaled to handle them.
- Solution: Simulate different peak load scenarios during testing based on user behavior assumptions. Regularly update and refine your stress tests as more information about user activity patterns becomes available.

4. User feedback on usability

• Blind Spot: While the functional requirements are clear, it is not clear how intuitive and user-friendly the system will be for teachers, students and administrators. Usability issues may become apparent late in the project.

• Impact: A poor user experience can lead to low adoption, frustration and a negative perception of the system, even if it technically works as expected.