MySTEMGrowth Survey Tool

Design Document

Sdmay 26-27

Dr.Rover

Dr. Rover, Yiqi Liang

Ethan Buenting: Backend/Cloud

Ethan Van Caster: Developer

Nina Gadelha: Cybersecurity Developer/Testing

Ryan Mamrot: Developer

Sam Craft: Developer/Testing

Caleb Hemmestad: Backend/Cloud

https://sdmay26-27.sd.ece.iastate.edu/

Revised: 09/30/2025

5 Testing

5.1 Unit Testing

- Backend service modules (e.g., Backend/api/UserNew/User.service.js (lines 20-134), Backend/api/surveyResults/surveyResults.service.js (lines 15-94), Backend/api/questions/questions.service.js (lines 16-18)) should have Jest test suites that stub the shared MySQL pool and verify each exported function: success paths, error propagation, and edge cases (missing parameters, zero results, constraint violations).
- Controllers such as Backend/api/UserNew/User.controller.js (lines 4-137) can be unit-tested with Jest + Supertest by mounting the router in an Express app and asserting status codes/payloads given mocked services and JWT secrets.
- On the frontend, utility hooks/components (iinspire-app/src/components/TheSurvey.tsx, iinspire-app/src/components/results/Profile.tsx) can be tested with React Testing Library + Jest DOM to ensure state transitions (phase changes, pagination, submission formatting) and derived text (card narratives) respond correctly to props/mock fetches.
- Where logic is mostly data transformation (e.g., grouping answers in TheSurvey), add pure helper functions and cover them with straightforward Jest assertions.

5.2 Interface Testing

- REST interfaces between frontend and backend (e.g., survey question fetch GET /api/questions, survey submission POST /api/survey-results, login POST /api/login) can be validated with pact-style contract tests: define OpenAPI schemas or JSON Schema expectations and ensure both consumer and provider tests stay in sync. Tools: Swagger/OpenAPI for spec, Jest + Supertest + jest-openapi for server response validation, and Mock Service Worker (MSW) for client-side interface simulations.
- Composition tests should exercise flows where multiple modules interact, such as user creation (frontend form → User.controller → User.service queries) or survey submission (React state → formatting payload → surveyResults.service normalization). Write Cypress component tests or Playwright interaction scripts that mount subsets of the UI with MSW stubs to assert full request lifecycles.

5.3 Integration Testing

- Critical paths include:
    - Authentication & Authorization: POST /api/login issuing JWT tokens (requirements: secure access by role). Integration suite should spin up the Express app against a test MySQL (or dockerized DB) populated with seed data, then run Supertest flows to ensure tokens gate endpoints like /api/usersprogram.
    - Survey ingestion and analytics pipeline: fetching questions, filling the survey (TheSurvey) and persisting results via surveyResults.service before the dashboard (Profile) renders normalized data. Cypress/Playwright end-to-end tests can automate a student logging in, completing the survey, and verifying stored results via API assertions.

    - Program management for coordinators/admins: API endpoints for program CRUD and student listing (/api/usersprogram, /api/newprogram, /api/program-survey-results). Integration tests should verify role-based access and data joins across Users/Program/SurveyResults tables.
- Tooling: Docker Compose (MySQL + backend) for consistent environments, Jest + Supertest for API integration, Cypress or Playwright for browser-level verification against the Next.js app.

5.4 System Testing

System level testing varies depending on what stage of the application pipeline is being tested. For frontend functionality changes, the changes are tested on a branch from within Android studio before being merged and implemented onto the main host. Tools included within Android studio allow the team to see and experiment with the elements as they are working on them, as well as test how different visual or logic changes affect the flow of the user experience. For backend testing, tools like Postman are used to ensure that endpoints deliver expected results, ensuring that critical functionalities like login and survey results are reliable and functioning properly. For changes or reconfiguration in hosting, multiple instances of the application are used. For example, when adjusting resource constraints and utilization or uploading new application versions within the cloud server, a 2nd instance of the application is launched. It is only when that 2nd instance is tested and validated that the URL is pointed towards the new instance and the previous one is taken down. This style of testing ensures that the application has little to no downtime when implementing new instances of the application.

5.5 Regression Testing

We work to ensure that new additional features do not break old functionality by using standard software practice, as well as version testing and control. When implementing new features, we work off of branches, individually adding features and testing functionality. Before any of these branches are merged, we ensure that we create new, independent functions to provide any additional functionality we need to implement said new features. Each of these new functionalities are tested on their individual branches before being launched onto the main platform. In the case that functionality critical to the program made it past testing, we are able to roll back to a previous working version of the application.

Critical features to the application for students include basic functionality such as sign in, logout, ability to take the survey, and ability to view results. While the look and feel of such functions has changed slightly and may continue to be tweaked, the underlying logic has remained the same, and any future changes will undergo thorough testing before any attempt to launch new versions. Similar practices will be implemented for any changes made to Admin or Program Coordinator roles. Critical functionalities for these roles include ability to add or remove members from programs, view program information, and export program data. While updates may be made to these functionalities, each change in functionality is rigorously monitored to ensure that these critical functions are operations at all times.

5.6 Acceptance Testing

To ensure the tool meets all functional and non-functional requirements, acceptance testing will be carried out in collaboration with our client. We demonstrate completed features in

meetings, walk through the application with the client, and validate that each component behaves as intended. During these sessions, administrators, coordinators, and participant workflows will be tested, as well as creating groups, distributing surveys, completing a survey, and viewing results to confirm the tool aligns with the operational needs.

5.7 Security Testing (if applicable)

Since we received an 'inprogress' system, the first thing we need to do is test the security of the current state of the website. From there, we can note possible vulnerabilities, research solutions to improve security flaws, design/implement solutions, deploy changes, test the new measures to ensure they fix previous vulnerabilities and refine implementation(s) if needed. Following this process will allow us to continuously improve the security state of the system and ensure our designs function properly, in order to keep user data secure.

Process:

- Test the current state of the system
    - Before implementing any changes, test misconfigurations, vulnerabilities and exploits to find weak points in the system.
- Note vulnerabilities
    - Record any security flaws found
        - Found examples: unprotected endpoints, plaintext sensitive data, weak password practices, etc.
- Research solutions
    - Brainstorm/research techniques in order to fix found security vulnerabilities
        - Future improvement examples; bcrypt() for hashing, implementing token, designing middleware, etc.
- Implement solutions
    - Make changes in order to fix found security flaws in respective areas of the project (frontend/backend coding, cloud settings, security configurations, etc.)
- Deploy changes
    - Push solution creations to Git and rebuild the site.
- Test new security measures
    - Test misconfigurates, vulnerabilities and exploits used before in order to see if the problems have been patched.
- Refine implementation
    - If vulnerabilities still persist, refine implementation until security flaws have been fixed.

5.8 User Testing

We intend on asking our clients to walk through the system and perform the tasks they intend to use in the future. As we make substantial improvements, we will present the model to our clients and ask them to demo use the system as they intend to in production. Not only will we be able to obtain feedback regarding the design of our changes, but we will also have our target user, put the system to use and ensure it functions correctly.

Periodically presenting our system and allowing our clients to use it, will ensure we are always testing the tool. Although each team member will be testing changes made every time they contribute to the project, having our clients test the product will bring a new perspective to testing.

Although it may technically work/function correctly through our eyes, the clients may have had a different functionality result in mind. They will also be testing system features in an order that will mimic functionality after deployment. This can lead to possible new vulnerabilities or system flaws discovered. Having them walk through the system, will allow us to observe and make changes based on their feedback. Having different perspectives test the system can also lead to new design flaws discoveries. The more we test the system before deployment, the more fixes we can make before the system is used in production; reducing future discovery of problems.

5.9 Results

What are the results of your testing thus far? Include any numerical, graphical, or qualitative testing results here? How do they demonstrate compliance with the requirements or addressing user needs? Use a summary narrative to discuss what you've learned and what next steps need to be taken.