5 Testing

Testing is critical to GridAI's development lifecycle due to its role in real-time power grid operations. Our testing strategy is closely tied to the project requirements, emphasizing early, continuous, and comprehensive validation across all components and interfaces. Tests are designed to reflect both technical correctness and real-world usage.

Some challenges include:

- Verifying synchronization of real-time collaboration features across multiple users
- Verifying visual accuracy for components like single-line diagrams and energy dashboards
- Simulating and validating diverse user workflows, such as residential users, energy providers, and educational users
- Testing performance and scalability during high-traffic or data-intensive scenarios

5.1 Unit Testing

The front end of GridAI is built using React, with many components forming the user interface. Unit testing focuses on validating these components in isolation to ensure they behave as expected under various differing conditions. We primarily utilized the Jest and React Testing Library to test our components.

These tools allow us to simulate real-world usage, validate DOM behavior, and ensure UI correctness without requiring a live backend.

Focus areas include:

- Validating component props, ensuring they render the expected output
- Verifying internal state management, including user interactions and context updates
- Ensuring expected UI behavior across different user roles and data inputs
- Testing custom hooks for connection logic and event handling

Tools:

- React Testing Library for rendering components, simulating interactions, and asserting DOM output
- Jest for running test suites, mocking functions, and validating logic and hooks

5.2 Interface Testing

Interface testing focuses on verifying that communication between individual system components is reliable, consistent, and behaves as expected during user interaction and data exchange.

Interfaces Tested:

- Dashboard and Widget Communication
    - Verify that widgets are placed in their correct position within the dashboard layout
    - Ensure real-time updates between the dashboard and widgets are consistent
    - Check the handling of events for widgets, such as dragging, resizing, and deletion of widgets
    - Test if widget states are updated properly in the dashboard UI
- Frontend and Backend Interaction

    - Validate API requests and responses for creating, updating, and deleting users, dashboards, and widgets
    - Test proper handling of edge cases, including missing or invalid data
    - Verify status codes and error messages for failed operations
    - Ensure real-time updates are received from the backend when using WebSocket connections.
- Code Editor and Widget Connection
    - Confirm that changes made in the code editor are applied live to the associated widgets
    - Verify that syntax handling and error feedback within the editor interface
    - Test widget behavior in response to user logic injected via the editor
- Frontend and Firebase Authentication
    - Validate sign-up, login, and logout functionality
    - Test token handling and expiration logic from the client side
    - Simulate invalid tokens and unauthorized access to ensure proper redirection
    - Ensure user role affects access to certain pages and components
- Tools
    - Cypress
    - Postman
    - Websocket Test Clients

5.3 Integration Testing

Our integration testing focuses on the interactions between the system components to ensure seamless functionality and compliance. The following are the critical integration paths and the testing strategies:

- Real-time data updates are central to the user experience since our features rely on live data, such as widgets or live code editing. The system must ensure in displaying the correct data is displayed. The testing strategy includes:
    - The system must be able to handle high traffic scenarios to ensure WebSocket connections remain stable and reconnect properly after disruptions.
    - Data integrity verifies that incoming and outgoing messages carry accurate, timely data.
    - Widgets need to ensure that they update correctly in response to real-time events.
- Authentication needs to be secure, and this governs user access. The testing strategy includes:
    - Utilizing the Firebase authentication integration by testing the sign-up, logins, logouts, and password recovery flows.
    - Verification for users with different roles, such as user or guest, can only access the appropriate data and features.
    - Ensure sessions persist appropriately and are invalidated on logout or expiry.
- The dashboard and backend integration rely on the API backends for data display and control logic. It must ensure that the endpoints are working efficiently for the best user experience. The testing strategy includes:
    - Validating the API endpoints to ensure the correct status codes and responses.
    - Handling errors through simulating failures to verify proper user feedback and fallback mechanisms.
    - Performance monitoring will measure response times and throughput under typical and peak loads.
- The tools utilized for these critical paths include:
    - Real-time Data:
        - WebSocket test clients
    - Authentication:

        - Cypress
        - Firebase Authentication Emulator
        - Postman
    - Dashboard and backend integration:
        - Postman
        - Monitoring tools like Firebase Performance

5.4 System Testing

Our system testing strategy will focus on overall system performance and different user workflows:

User Workflow Testing (Automated using Cypress): This ensures every user interaction within the GridAI platform works as intended. The focus is on end-to-end scenarios that would mirror real-world use cases:

- DSO (Distribution System Operator) Scenarios:
    - Test workflows for grid operators managing power distribution, such as voltage stability or balancing load across regions, ensuring our UI updates correctly and data remains accurate during the test.
    - This is important because DSOs rely on precise, real-time data to make decisions and maintain grid reliability.

- DERA (Distributed Energy Resource Aggregator) Management Workflows:
    - Validating workflows to aggregate and manage distributed energy resources (i.e, battery storage, solar panels, etc) to optimize energy sales or grid contributions.
    - This is important because DERAs need streamlined processes to manage multiple energy sources, and our testing will help ensure the reliability for users trying to profit from energy.

- ISO (Independent System Operator) Monitoring Procedures:
    - Testing to ensure ISOs can monitor grid performance, track energy consumption, and respond to responsive energy demand spikes or other signals through our platform.
    - This is important because ISOs oversee larger-scale grid stability, so GridAI must provide reliable, responsive, and accurate information.

Performance Testing (Automated using Lighthouse): Provided the data-intensive nature of GridAI, performance testing is critical to ensure the platform remains responsive and stable during heavy usage:

- Load testing with multiple data points (tens, hundreds, thousands, etc):
    - Simulate thousands of data points (real-time energy readings, historica,l) testing for multiple nodes being processed on the system (MapBox highlighting functionality)
    - This is important because our entire platform was built and relies heavily on the functionality of processing large datasets quickly; slowdowns could frustrate users or delay critical decisions for important users.

- Resource Utilization Tracking:
    - Monitoring server CPU, memory, and database usage during situations of scalability, ensuring GridAI scales accurately and efficiently.
    - This is important because efficient resource use helps keep our platform cost-effective and reliable for each user.

- Response Time Monitoring:
    - Measuring how quickly the platform responds to multiple user actions/interactions, such as loading a dashboard/widget or calculating real-time node power usage.
    - This is important because our users expect our application to work with instant or near-instant information and feedback from their requests, especially during peak energy usage or time-sensitive tasks.

Multi-device and Multi-browser Testing (Automated using BrowserStack): The system must work consistently and efficiently across devices and browsers that our users could be using.

- Mobile Responsiveness:
    - Testing that our platform is mobile-friendly, and is adapting to smaller and vertical screens, with proper layouts and displaying data optimally.
    - Ensure touch-based interactions (swiping through different screens, dashboards, and button interactions) are smooth and intuitive for the user.
    - This is important because many users could be accessing our platform while on the go and expect a desktop-like experience without compromise.

- Chrome/Firefox/Safari compatibility:
    - Verify all features (node data visualizations, transactions, etc) render correctly and function identically across major internet browsers.
    - This is important because users may have their own browser preferences, and we must mitigate inconsistencies as much as possible.

5.5 Regression Testing

To ensure our additions do not detract from the functionality currently found in GridAI and that new functionality is implemented correctly. This involves continuous regression testing and re-running previously validated components we are changing. This must be through test cases whenever updates are made to these components. Many tests are similar, if not identical, to previous testing formats, but are still essential to do after changes.
Critical features to test:

- Data visualization tools - Includes the live and historical data present in the Map Box, Widgets, and Market Dashboards.
- API integration stability - Validating all new and existing uses of the backend API in the front end and ensuring they are seamlessly integrated.

- Responsive Design - Testing all UI components to ensure they are easy to use, understand, and perform without error.
- User validation - Validating that user types get sent to the right page. This is especially important in the market dashboard and map, where separate user types see entirely different information.

Testing Approach:

- Requirement-driven testing -
- Automated regression suites - Cypress and other test suites can automate different aspects of regression testing.
- Manual verification - Mostly for API validation, directly using the backend with test calls using systems like Postman.
- CI/CD integration - Integrating CI/CD pipelines could give us automated regression testing, an elegant way to lighten the load of manual testing.

Tools and Metrics:

- Postman - Manual backend API testing allows us to see the intended outcome of an API call and see if the front end uses it appropriately.
- Cypress - A testing service that can be manually used in a browser or automated into a CI/CD pipeline.
- Lighthouse - One of the most common UI responsiveness trackers on the market. Allows the tester to see a rating based on responsiveness, accessibility, SEO, and best practices.
- Web vitals - Another of the more common UI responsiveness trackers, reporting on how well the software performs.
- Jest - An automated testing service widely used in JavaScript and TypeScript projects.
- Copado - Another automated testing platform using CI/CD to test after every change. However, it has a potentially high cost associated with it.

5.6 Acceptance Testing

Our acceptance testing strategy will ensure that both functional and non-functional requirements outlined in our design document are fully met. The focus is on verifying that the system from the end-users' needs to verify the readiness of the product for deployment.

Client Collaboration:

- Demonstration Sessions - Hosting walkthroughs of major testing features using real-time datasets, allowing the client to observe workflows and provide immediate feedback.
- Client Test Cases - Incorporating test cases provided by clients to validate the real-world cases
- Feedback Loops- Collecting client feedback from test client test cases and addressing concerns and growths that can be applied

Functional Requirements:

- Scenario Testing - Using real-world scenarios for testing cases across user types
- Checklist Confirmation - Using a structured checklist based on requirements to keep track of confirmed scenarios
- Hierarchy Access - To confirm that every type of user has the proper permissions for the application

Non-Functional Requirements:

- Performance - Confirm that the application is reaching performance goals and benchmarks while it is running
- Accessibility - Making sure that the application is usable across multiple platforms and devices
- Reliability - Test the application for extended periods to ensure that it can remain stable

Tools:

- Postman - Validate workflows and data through API testing
- BrowserStack - Verify that the application runs on multiple platforms of devices
- Client Feedback - Take in client feedback and apply it to our testing and application
- Cypress - Use this for functional testing

5.7 Results

We are still in active development, but several testing efforts are already underway. For unit testing, we are finalizing the CI/CD pipeline and creating test cases for each component so that code cannot be merged unless it meets our baseline criteria. On the interface side, we have verified that the frontend and backend communicate correctly and that the frontend authenticates with Firebase as expected. Work is ongoing to ensure the code editor reliably updates its associated widgets. For integration testing, we use Postman after each major change to confirm data flows smoothly and remains consistent between the frontend and backend. System-level testing has not started yet, but we plan to capture the performance metrics needed to confirm the application is fast and reliable. Regression is covered by shared Postman collections that we run after every sprint, and we are expanding those tests to guarantee new updates coexist with existing code. Finally, full acceptance testing will happen once the application is more mature and real data is available.