5 Testing

The team is implementing a robust testing framework that covers both individual components and the system as a whole. The system consists of three main parts: the hardware in the parking lot (Raspberry Pi, ESP-32s, cameras, etc.), the server hosted by Iowa State, and the user application.

Hardware Testing Strategy and Summary:

For the hardware, the team aims to ensure that the Pi and its dependent components operate at full capacity under all conditions and have proper procedures in place for system failures. Each component will undergo extended full-load testing—for example, running the CV model on the Pi in a hot environment with high-resolution video input. After stress testing individual subcomponents, the same approach will be applied to the full system. This process should help identify potential failure points and guide the development of handling procedures for issues such as power loss, memory corruption, and more.

Server Testing Strategy and Summary:

The backend must be secure and reliable as it contains fragile data. To do this, there must be strenuous testing on the server to ensure availability and functionality. A few ways of doing this include: using a testing framework, Mockito, generating a code coverage framework, and straining the security via penetration tools.

Application Testing Strategy and Summary:

Software testing will be handled by unit and integration testing that test all of the application modules as implementation is completed. This process will ensure that possible edge case complications with even the most simple of application components are covered by tests to allow for easy application use. Integration tests will be run to ensure the React components within the application are successfully communicating with the live server and data is being properly processed. Finally, acceptance testing with the client and back end team will ensure that all the necessary features are implemented with expected functionality.

5.1 Unit Testing

Hardware Unit Tests:

Check for transmission of pictures while evaluating latency and resolution. This test will be performed in a range of temperatures, brightnesses, and ranges.

CV Efficiency:
Evaluate the computer vision model's accuracy across various video inputs, including high and low visibility scenarios with differing traffic densities.

Temperature:
Ensure the Pi's components remain within safe operating temperatures (throttling begins at 85°C). The CV model will be run at different ambient temperatures—room (72°F), hot (100°F), and cold (32°F)—while monitoring the CPU and AI accelerator to identify any thermal issues.

Connectivity:
Test the Pi's ability to connect to the school's network by streaming data from the parking lot to the server over an extended period. The hardware team will monitor data loss, connection speed, and dropouts.

Database Logic:

To make sure the backend code is storing and fetching correct results to ensure correct behavior for the front end.

Software Unit Tests:

The front end team is using component level testing to prevent any unexpected behavior or micro failures within any React modules. The team is using the React Testing Library to ensure the user interface is functioning properly and plan to use Jest to test the JavaScript behavior.

5.2 Interface Testing

Hardware System Interface:

The hardware system consists of several components: the Raspberry Pi with an AI accelerator, multiple ESP32s, and cameras connected to both the Pi and the ESP32s. The Raspberry Pi serves as the central node, sending processed information to the server, while the ESP32s and their connected cameras relay video back to the Pi for processing. The ESP32s send pictures to the Pi via HTTP protocol. This connection will be tested by continuously sending pictures from the ESP32s to the Pi in a variety of conditions (temperature range, power outages) while verifying data integrity and connectivity.

Hardware to Server Interface:

The Pi will stream data related to detected cars and license plates to the server for processing and integration into the application. This data will be sent in JSON format. To test accuracy, a collection of pictures with known data points will be given to the Pi's computer vision model. Then, the expected results will be compared to the actual values received from the hardware. This approach also allows for scaling the number of inputs to match the expected number of cameras in the parking lot.

5.3 Integration Testing

ESP32 and its Camera to Pi Connection Integration:

This integration is critically important as it allows for scalability within the system. The connections between the ESP32s and their cameras allow us to monitor all the spots in the lot. This integration will be tested by incrementally increasing the number of ESP32 nodes connected to the Pi and streaming video at each point, monitoring the integrity of the data being streamed in the mean-time. The team will then scale the integration by adding additional nodes (ESP32s and their respective cameras) and reconducting the test.

Hardware Server Communication Integration:

For the application to work, the hardware team needs to connect the hardware that collects information from the Pi to the server. This serves as a critical integration point in the system. The data will be formatted according to the specifications of the software. The team plans on testing this integration by running the model on video input with known values and then cross-checking the results with them (this was outlined above).

Server to Application Integration:

The front end team has already run integration tests with the account registration process, where the application cross references the user-inputted email address to ensure validity and then processes a POST

request to the Spring Boot API on the live server. This integration was tested with the Login process where the application checks the user-inputted fields and returns a valid, already registered user account that is pulled from a GET request from the live server. The front end team also used a dummy parking lot entity with a set remaining parking capacity enum value in the back end to pull the capacity field from and display it on the Map Page upon selection of the specific lot. When the Expo application pulls the capacity data, it checks the spot's "status" field and then calculates the number of available spots. Future front end integration tests will be handled in similar fashion where the pulled and pushed data to the live server is manipulated and compared on the application-side.

System Testing

In order to conduct full system tests, the user experience will be emulated. When the full system is assembled, there may appear edge cases that aren't considered in development (a license plate is improperly mounted, the user has an outdated phone, etc.). The system tests will be updated to account for these cases, resulting in a robust testing process as the product is used.

The stress testing on the individual system components will continue to be tested while the whole system is running (temperature testing for the Pi, heavy user requests from the application, etc.). This will avoid unexpected outages.

5.4 Regression Testing

Unit tests cover software regression testing. Running the same unit tests as the software changes ensures there are no potential regressions as the front end team continues to develop code. Although tests can be updated and added, running all the tests before deployment ensures all functionality remains working.

The hardware is responsible for capturing the most important data for the main process of this project, so regression testing with the hardware implementation is crucial to the team's success. By checking a collection of tests before deploying the code to the Pis, all the features can be verified. Also, running the same Docker container on all devices allows new device types to be used. When new devices are added, the old devices will still be tested before deploying.

5.5 Acceptance Testing

The acceptance testing will be handled directly with the client during weekly meetings. As the team continues demonstrating new features and implementations, the client will decide whether the changes are acceptable and meet his vision for the project. Furthermore, requirements have been discussed at the beginning of the project which will continue to be updated according to the client's concerns.

5.6 Security Testing

By using Iowa State's network for the server and Raspberry Pi, the network will follow Iowa State's standards. This includes a reverse proxy and authenticated logins. The ESP32 nodes are connected to the Pi using a local WiFi hotspot (without an internet connection). This isolates the nodes from Iowa State's network.

Additionally, the security of any Docker container being used can be analyzed using Docker Scan. Since all backend code as well as Raspberry Pi code is running within docker containers, all the project's code can be scanned using Docker libraries.

Because of these protections, Docker and Iowa State assume the burden of testing.

5.7 Results

As the early version of the project progresses, the unit and integration tests that the front end team are deploying ensure that the application is ready for the next step. The team has run unit tests on the Login/Register with input handling and analysis. The case-sensitivity unit tests failed and the team made the proper adjustments to meet the spec requirements regarding this edge case. Another integration test that caused a design adjustment was the failure to navigate to the Reservation page prototype when the app was unable to fetch parking lot capacity data from the live server on the Map Page. The team has learned of this unique case and is currently working on the necessary debugging steps to fix this error and has plans to communicate it to the client.

The backend code has CI/CD deployment setup. Because of this integration, deploying code and debugging has been much more efficient. By verifying the code builds before it is sent to the Raspberry Pis, issues have been caught early. Going forward, functionality tests will be included in the CI/CD process.