Availability Prediction Based on Multi-Context Data

FINAL REPORT

Group: sdmay19-33

Client: Goce Trajcevski

Team members:
Justice Wright - Report Facilitator
Shane Impola – Scribe
Noah Chicchelly – Meeting and Communications Facilitator
Nick Schmidt – Software Systems Engineer
Tristan Anderson – Network Systems Engineer
Brendon McGehee – Hardware Systems Engineer

Team email: sdmay19-33@iastate.edu

Team Website: https://sdmay19-33.sd.ece.iastate.edu

Version 1.2
Revised: April 22, 2019

0 Executive Summary

Presently restaurant wait time prediction is done by archaic methods largely based on intuition and personal aptitudes. Wait times and their accuracy can largely influence the likelihood of a customer to return, or stay in the first place. This report describes a project to use a small IoT device with an accompanying mobile application to use sensors and data analytics to more accurately and reliably convery restaurant wait times to potential customers. This basic system consists of a sensor node machine, an aggregator for the nodes, a backend hosted in AWS, and a mobile application. Through the use of this sensor data this project aims to use human interaction and tendency to predictively model wait times. We recommend implementing this system on a trial basis, and openly asking for feedback from customers. If the trial proves successful we recommend working towards full-scale integration and replacing existing wait-time systems.

1 Requirements Specification

A design does not fully succeed unless it meets every need that is required of it. That is why before implementing our design we took careful stock to note our problem, assumptions, and requirements. Our findings are detailed below.

1.1 Problem Statement

Currently, waiting times in restaurants often feel up in the air. The host/hostess will often estimate the amount of time you will have to wait for a seat and leave you to wait hopelessly for your buzzer to alert you to an available seat. We use several untapped methods of data collection to provide an accurate and readily available wait time estimate for potential customers and also for those customers waiting for a seat.

To accomplish this goal, we plan to install sensors into the seats of a table. Using data on how many occupants a table has, how long they have been there, and data from similar situations, we will accurately estimate how much longer it will be for a table to open up. By using these data sources which are currently unavailable or difficult to track, we will vastly improve the predictions to wait time and make them available to customers via a mobile application.

In addition to wait time predictions for customers, we allow owners of restaurants to look through our data. In doing so, owners can see what tables are most preferred, the average party size attending their restaurant, how long people stay for, and other similar data. This information could allow owners to improve the dining experience for customers, boosting potential sales.

1.2 Assumptions and Limitations

Assumptions:

The final product will only be implemented indoors - the hardware of the project does not need to stand up to any harsh weather conditions.

Hardware component will not have any power limitations - the hardware component will have access to electrical outlets.

Limitations:
Sensors need to be unnoticeable inside a restaurant - The sensors used to collect data need to be implemented into a restaurant setting without impeding the normal operation of the restaurant.
The application needs to be usable by all types of technical backgrounds - The app can not be overly difficult to use because people of all technical backgrounds will be utilizing the application.

1.3 Functional and Non-functional Requirements

Requirements fundamentally drive the design process. We found the following requirements to be necessary of our solution and they served as the principals we built our design upon:

Functional Requirements

- Sensor nodes must be able to continuously and accurately detect the occupancy status of a seated area
- A microcontroller must continuously check these sensors and relay sensor events downstream to a centralized hub or database
- Analytical algorithms must continually process sensor data into human readable and accurate (defined later in testing) wait times
- Mobile application must retrieve and display wait times as well as information relevant to user-group of client

Non-functional Requirements

- The data must maintain high availability as the information stored may be relevant at any time depending on the client
- Data integrity and security must be maintained, as breaches could be detrimental to businesses
- Our model needs to be highly scalable to work in all sizes of restaurants
- The app should maintain a high level of ease in usability, as not to be cumbersome to incorporate

2 System Design & Development

With our problem thoroughly scoped, and requirements known we were ready to make design considerations based on that input.

2.1 Design Objectives and Constraints

Before approaching the design stage of development it was important to have an understanding of the guiding principles, objectives, and constraints of which our design would need to conform.

Firstly, our functional and non-functional requirements as outlined section one of this document gave us solid direction.

From there we expanded these requirements into ideals that would ensure the design adhered to these requirements. These ideals included but were not limited to:

- **Durability**: A self-contained durable enclosure to house the sensor node boards would both lean towards higher system availability, as well as put less strain on customers for maintenance.
- **Scalability:** Our solution needed to be able to accomodate and enhance any size of business.
- **Reliability:** The nodes and backend need to be constantly available with predictable behavior. Downtime or inaccuracy is a loss of data, which is as good as a loss of revenue to potential customers.

2.2 Design Plan

With these guiding principles in mind our design took shape. The prototype plan, pictured below, consisted of three simple subsystems to both capture, analyze, and deliver data from the table to the customer in real time. Those subsystems included the data collections system (sensor nodes), the data analytics system (backend databases/algorithms), and the client application (mobile app).

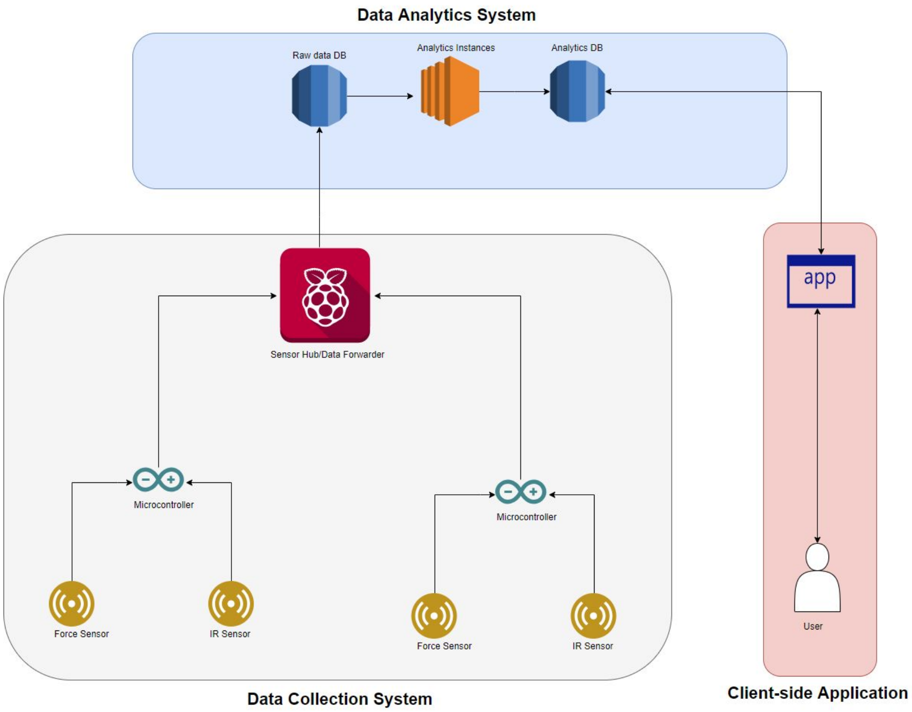
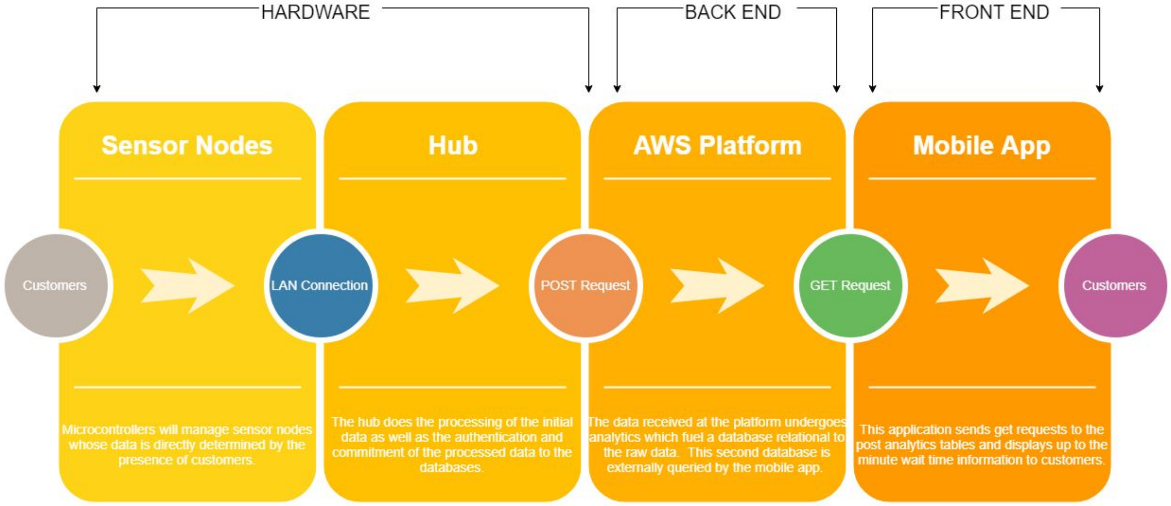


Figure 1: High Level Overview

While the individual components of the subsystems would be altered as detailed in the following subsections, the core of this three system design remained intact through the final result.

2.3 Process Flow

To best understand our design we outlined the system (figure 2) through the lens of the information flow process. As you can see, the process is initiated by the customer interacting with the sensor nodes, who then communicate with the hub. The hub aggregates all the information from the nodes and sends them to our databases hosted in AWS. After internal analytics are performed the mobile application receives the output from the AWS Platform and conveys the information to the customer.



Microcontrollers will manage sensor nodes whose data is directly determined by the presence of customers.

The hub does the processing of the initial data as well as the authentication and commitment of the processed data to the databases.

The data received at the platform undergoes analytics which fuel a database relational to the raw data. This second database is externally queried by the mobile app.

This application sends get requests to the post analytics tables and displays up to the minute wait time information to customers.

Figure 2: Information Flow Process

2.4 Use-Cases

Development is largely centered around use cases. Without understanding your users and their individual needs a project cannot truly be sure it is being successful at providing all desired functionality. After identifying our users as both restaurant customers and restaurant employees, we developed the following use case scenarios:

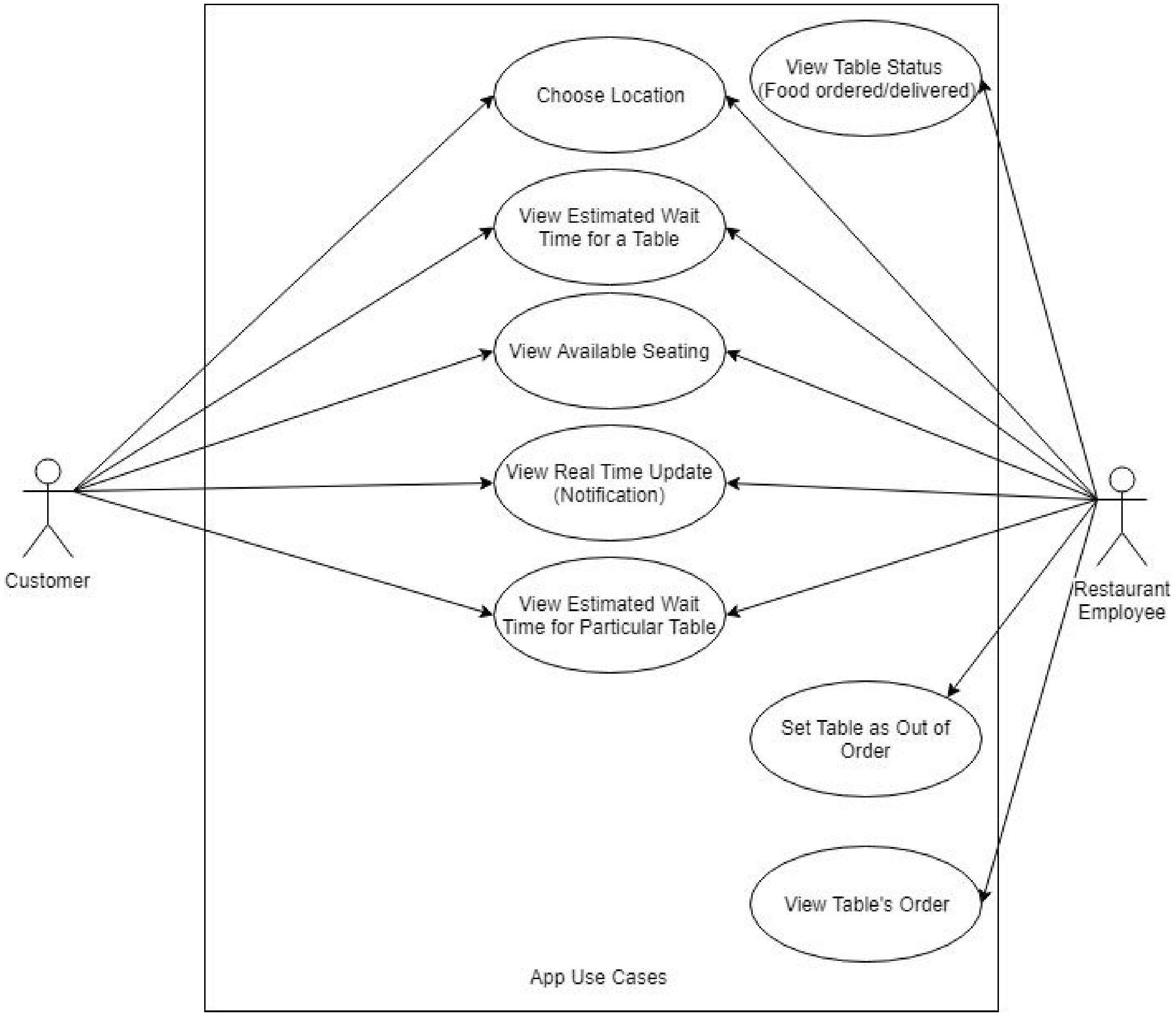

Choose Location
View Table Status (Food ordered/delivered)
View Estimated Wait Time for a Table
View Available Seating
View Real Time Update (Notification)
View Estimated Wait Time for Particular Table
Set Table as Out of Order
View Table's Order
Customer
Restaurant Employee
App Use Cases

Figure 3: Use Case Diagram

Name	Description
UC1- Choose Location	Choose a location from the list of available locations.
UC2- View Est. Wait Time for Table	View the wait time to receive any table at the given restaurant.
UC3- View Available Seating	View all of the seating for the restaurant and see which seats are available.
UC4- View Real Time Update	Receive a notification on the phone when a table becomes available.
UC5- View Est. Wait Time for	View the wait time of a specific table on the seating chart.

Specific Table	
UC6- View Table Status	View the status of the current table, whether it has ordered food, is currently eating food, has received check, ect.
UC7- Set Table as Out of Order	Set specific tables as out of order, making them unavailable for view in the app.
UC8- View Table's Order	View the food that a specific table has ordered.

Table 1: Use Cases

With these use cases in mind we were able to move into the implementation phase.

3 Design Implementation

With a firm understanding of how our requirements shape our design we were ready to move from design to system implementation.

3.1 Resource Selection

Before implementation can take place it's paramount that we understand what tools we plan on using, and more importantly why those are the tools we are selecting. The following sheds insight onto our resource selection process and conclusions.

Sensors: Sensor selection was a very important task for this project. Sensors needed to be able to simultaneously meet our functional requirements, while occupying the least amount of space, and delivering the most accurate data possible with the least amount of sensors, given the constraints of their setting.

We ended up selecting a force sensitive resistor in conjunction with an infrared sensor. Multiple factors led us to this decision. We knew we wanted at least two sensors that acted on different stimuli as to best eliminate false positives. The weight sensing and proximity sensing abilities that each sensor provided, offered just the diversity we were looking for. Additionally, through the passing of our hardware tests (outlined in section 4.1) we were able to validate that they were accurate, repeatable sensors for the type of information we wished to collect.

Development Tools: We used a multitude of tools in the development of the software component of this project, to include Amazon AWS, Flutter, and Arduino products.

Amazon AWS was selected to host our backend implementation. It was selected as it is the industry standard platform hosting environment, and offered many well documented and versatile tools to our environment. With a long history of being a leader in platform development, this decision was relatively simple to reach.

Arduino products were used as both microcontrollers at various points, as well as IDE's to test small snippets of code with. This decision was also relatively simple, as our hardware team had much experience with Arduino products, and they, much like Amazon, have become known industry leaders/standards for non-proprietary devices.

Lastly we came to the decision to use Flutter for our mobile development. This decision was the most difficult in our resource selection process, as Flutter is not nearly as notoriable or influential in the mobile development field as the other tools we used were in their respective fields. The main factor that led to the selection of Flutter as a development tool was the fact that it allows for simultaneous iOS and Android development. Since our product needs to appeal to anyone who could work or eat at a restaurant it was important that we reach as wide of an audience as possible. Since we had a small mobile team who had experience with Flutter, this seemed to be the most time effective solution to reach both markets, without expanding the team and splitting their efforts on dual developments.

3.2 Full Implementation Model

With our tools selected the next step was implementation. With our tools, design limitations, and requirements in mind, this model conveys our final implementation:

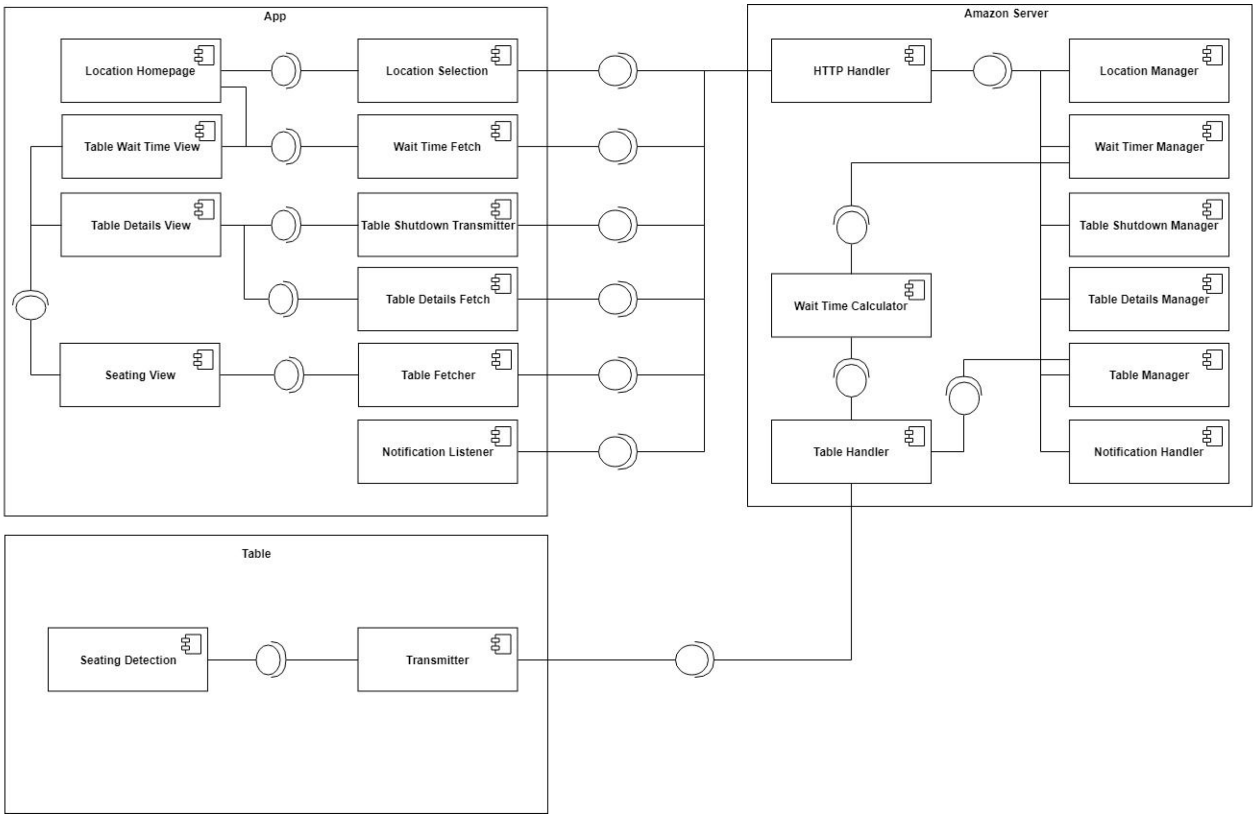


Figure 4: Component Diagram

Application:

Component	Description
Location Homepage	The launch page of the app, starts with a location selection and update to the wait time for a table and navigation to other pages.
Location Selection	Communicates with the server to get the available locations.
Table Wait Time View	Holds the GUI and interactivity for seeing the wait time of a specific table.
Wait Time Fetch	Communicates with the server to fetch the wait time for a table or

	restaurant.
Table Details View	Holds the GUI and interactivity for viewing the details of a table (food ordered, how long the customers have been there, what stage they are in), also contains GUI controls to set table as out of order.
Table Shutdown Transmitter	Communicates with the server to notify the server when a table should be set as out of order.
Table Details Fetch	Communicates with the server to retrieve the details for a given table to displayed in the details view.
Seating View	Contains the GUI and interactivity for the seating of the restaurant. Allows users to change into the wait time view, and if the user is an employee, the table details view.
Table Fetcher	Fetches all of the information about tables available at the restaurant to be displayed in the app.
Notification Listener	Listens to notifications from the server to notify the user when a table becomes available.

Table 2: Application Component Description

Amazon Server:

Component	Description
HTTP Handler	Listens for HTTP requests and route them to the correct manager.
Location Manager	Retrieves information related to the location.
Wait Time Manager	Retrieves information related to the wait time.
Table Shutdown Manager	Shutdown or open specific tables for the given restaurant.
Table Details Manager	Retrieves detailed information about the tables (food ordered, current status).
Table Manager	Retrieves information about the tables in the restaurant (locations, seating available, location).
Notification Handler	Setup a notification line for the app, notifies the app when a table or a specified table becomes available.
Wait Time Calculator	Calculates the wait time for a given table.
Table Handler	Listens for transmissions from table transmitters and pass the data to

	the correct managers.

Table 3: Amazon Server Component Description

Table:

Component	Description
Seating Detection	Detects whether someone is currently sitting in the seat
Transmitter	Transmits relevant data.

Table 4: Table Component Description

4 Testing and Evaluation

While design implementation is important, testing is what ultimately determines if the implementation is correct, valid, or even feasible. Below are some of the testing criteria we used to judge our system at large and low levels alike, as well as the results we experienced from each set of tests.

4.1 Hardware Tests

A large portion of our project was dependent on the sensor nodes and the data they provide. This is the top of the funnel as far as our information flow is concerned, so a considerable amount of testing attention was given to the hardware subsystem.

FR.1: This is a test program for the an Arduino Nano that prints out sensor values in real time to test if the sensors function properly.

Test Case: Test if the sensors are outputting data as expected.

Test Steps:

1. Start the test program.
2. Manipulate the sensors by hand.
3. Watch the printed output and determine if they are as expected.

Expected Results: The expected printed output value should remain within 5% of the physical changes in the state of the sensor.

Actual Results: Sensors worked predictably and within a 5% tolerance of expected values for objects at rest. It was determined that customers are not likely to be moving excessively while eating, so this was deemed acceptable passing criteria for this test.

FR.2: This is test software that takes a set of raw input from the sensors and processes it in order to determine if the microcontroller is processing sensor output properly.

Test Case: Test a set of data on both the microcontroller and a computer.

Test Steps:

1. Run the arduino program, enabling a debug option that sends all inputs to the computer.
2. Simulate the data processing on the computer.
3. Compare the result of the arduino program to the simulation.

Expected Results: The output values from the microcontroller should be within 5% of the simulation.

Actual Results: The output values were within the 5% allowance limits. This was deemed as a passed test.

4.2 Server Tests

While our backend is important most of the testing around server systems were in regard to availability and accuracy of stored data. Testing for accuracy of information supplied to or from the server system was relegated to both the hardware and application tests.

FR.3: This is a connectivity test that pings the server and await a response to determine if the server is online and responding to requests.

Test Case: Test that the server responds to requests as expected.

Test Steps:

1. Send a request to the server test address.
2. Wait for a response.
3. Compare the response to a preloaded value.

Expected Results: The response from the server should match the preloaded value.

Actual Results: The response from the server matched the preloaded value and the test was considered successful.

FR.4: This tests the functionality of the database.

Test Case: Test that the database handles inputs and requests as expected.

Test Steps:

1. Send a request to the database test address.
2. Wait for the server to perform a set of automated queries on the database and respond with the result.

Expected Results: The result should indicate that the tests were successful, if they properly mirror the output expected for the supplied input.

Actual Results: There were some instances of failing this test during initial experimentation. It was continually revisited and reevaluated with this test until a passing grade was achieved.

4.3 Application Tests

Application tests consisted of both availability testing and accuracy testing of the analysis subsystem. An emphasis was placed on information availability, reliability, and accuracy.

FR.5: This is a set of test functions on the apps that implement the same functionality as described in tests FR.3 and FR.4 to determine if the app is communicating properly with the server. See those tests for details.

FR.6: Test the functionality of our predictions.

Test Case: Test the accuracy of our prediction algorithms against big data.

Test Steps:

1. Generate big data real-world simulations of our test environment
2. Cross the data with ours to make sure it's within an ~10% margin

Expected Results: The delivered post-analytics data should match our predicted post-analytics data, within a 10% margin of error.

Actual Results: Our analytics algorithm always met within a 10% margin of our expected output with test data, but had to be retested several times as we continually adjusted our algorithm, and thus had new expected data.

4.4 System Tests

Tests in the subsection may have had applications across multiple subsystems but did not mandate that all systems were working in full integration for testing purposes. These tests largely determine the viability of application within certain likely scenarios for our sensors or data.

FR.7: This is a general test to determine scope of our sensors.

Test Case: Test to determine the ideal weight range we should be sensing for.

Test Steps:

1. Test real-world data for a general range of weights that should be tested.
2. Verify we are only triggering at the minimum and maximum ranges.
3. Determine if we keep false positives below a threshold.
    a. If we do, test complete.
    b. If we do not, repeat and adjust min/max values.

Actual Results: We tested continually until we had data that for our sensors that satisfied the stipulations of this test.

FR.8: Test if the system works in a specific environment

Test Case: Test that the tables/chairs/location can house the required sensors.

Test Steps:

1. Test units (tables/chairs) for the weight requirement for sensor trigger.
2. Test functionality of sensors in tested locations, repeat until sufficient.

Expected Results: This should result in sensor placement yielding results within 10% accuracy of the expected value given the induced stimulus.

Actual Results: All environments tested yielded results well within 10% of expected values, and our initial calibrations were deemed applicable for use in most implementations.

4.5 System Integration Testing

FR.9: Full-scale integration testing of all subsystems.

Test Case: Test that all systems are able to communicate live data.

Test Steps:

1. Setup sensor nodes on a mock table environment.
2. With an empty database, start the mobile application.
3. Occupy a seat.
4. Observe if the data is both logged in the backend, and conveyed to the mobile device.
    a. If data is present at all stations, test is complete.
    b. If data is not present at all stations, end testing and retest when appropriate fixes have been made.

Expected Results: Data is sent to the backend and back through the app with less than a ten second delay.

Actual Results: There was initial error in communication between backend and the mobile application but after several retests communication was established to the standard outlined.

FR.10: Full-scale implementation in a live scenario.

Test Case: Setup our product in a live testing environment where it can record data, be observed by clients who can offer feedback.

Test Steps:

1. Find a potential location to test the system, and negotiate for the ability to use the facility.
2. Design a testing scenario that might mimic a use case specific to the client.
3. Place the system and instruct the client on how to use/observe the system.
4. Collect the system.

5. Listen to direct feedback and assess if any design adjustments are warranted.

Expected Results: Complete the test steps and at a minimum have a baseline measurement for the success of the project in a practical setting.

Actual Results: Finding a location that would both allow us and be perceived as beneficial given our implied use cases was challenging. We were unable to identify any restaurants that would allow our system to be implemented for a trial period. Eventually we were recommended and began conversations with the library. Talks for testing are still underway, but the data we would expect to collect there has versus a restaurant has led to some uncertainty in the success criteria we can outline, or what scenarios we need to engineer to make the test mutually beneficial. This is currently ongoing.

4.6 Evaluation

While several tests required retesting before a passing grade was achieved, we did manage to pass nine out of ten large scale documented tests, with ongoing progress on the tenth.

While some might point to the retesting as a sign of failure we view it as an indicating factor that we designed stringent testing criteria. The fact that we were able to make our design conform to our testing criteria, and not the other way around was crucial to the success of the final product.

5 Project and Risk Management

We approached this project under the Agile project management framework. This framework was selected for its easy integration with our required client and advisor meeting structures, our overall previous experience and satisfaction with the framework, as well as agile's ability to better direct work of several subsystem teams than competing models such as the waterfall model. Agile proved to serve us well over the course of the project, and would likely be our go to if we were to choose a management style again.

5.1 Roles and Responsibilities

Well defined roles are an integral part to project management and team success. Without established roles and associated responsibilities a team can end up with much wasted or misguided energy and the quality of their work suffers. These were the following roles we agreed upon for this project.

Justice Wright - Report Facilitator (Secondary Hardware Systems Engineer)

Shane Impola – Scribe and Primary Backend Systems Engineer

Noah Chicchelly – Meeting and Communications Facilitator (Secondary Backend Systems Engineer)

Nick Schmidt – Software Systems Engineer

Tristan Anderson – Network Systems Engineer (Secondary Software Systems Engineer)

Brendon McGehee – Hardware Systems Engineer

These roles were selected from both individual and group input. While these were the assigned roles it was also noted that everyone was a project member first and their role second, meaning if they had an idea or grievance with something outside of their assigned focus area they were encouraged to discuss it with the group/other team members as necessary, and could feel free to pitch in or recruit help elsewhere as needed. Thus, these roles gave us direction without the sense of limiting our effectiveness or unit cohesion.

5.2 Project Schedule

We prepared Gantt charts to outline our desired progress for our sprints. The originally proposed plans for first semester was as follows:

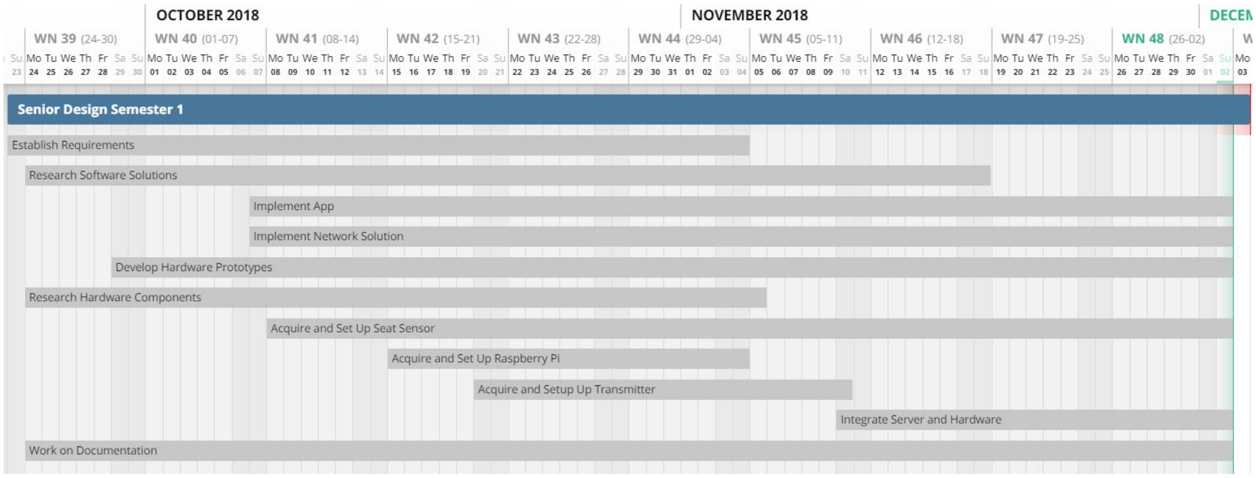


Figure 5: 491 Gantt Chart

The goal of this semester was to establish a functioning prototype based off of the design shown earlier in figure 4 . While a prototype that was able to convey subsystem integration was completed on this timeline, it did fall short of some of the sub-goals we had for the prototype. Primarily, full backend integration was not achieved, only partial. This did not change our second semester gantt chart much at that time, though in hindsight it likely should have. Our second semester plans were as follows:

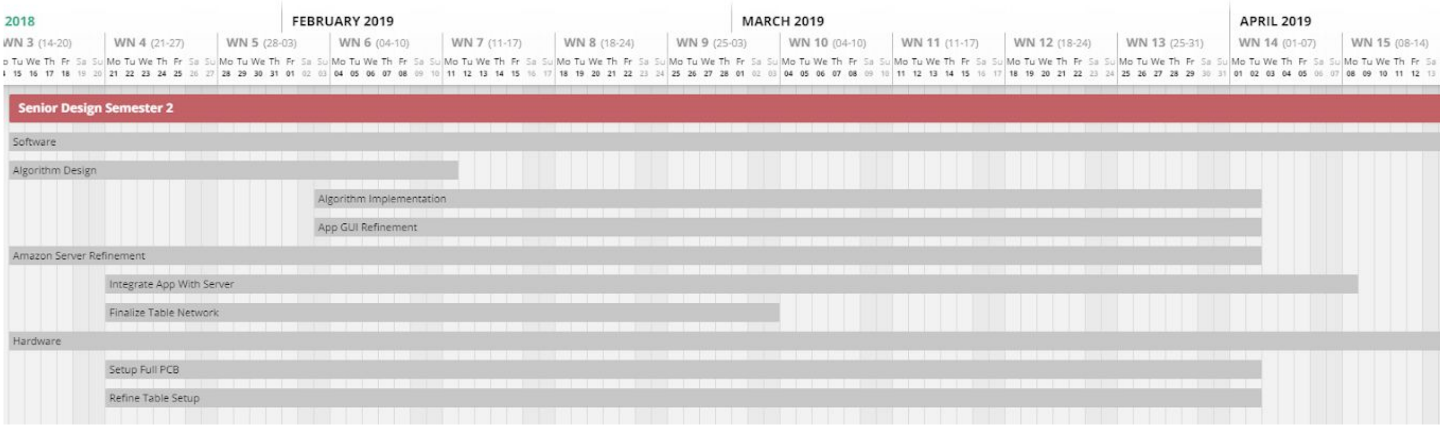


Figure 6: 492 Gantt Chart

While this timeline made sense and seemed perfectly in scope at the time, both new testing requirements as well as complications and early completions shifted things quite considerably from the original model. Most notably, the hardware components were completed several months ahead of schedule in late February, and the algorithm implementation was started earlier, and became the continual focus for the project. There was also some major redesigns and reconfigurations of the backend, which were unplanned but ultimately worthwhile.

5.3 Risks and Mitigations

During the course our project we did a pre-emptive risk assessment with accompanying mitigations based on the areas we most expected to encounter adversity during the project. As always, no plan is perfect and we experienced a mixture of these anticipated risks and entirely new risks during development. The following section details both types of risk and gives an overall summary of our experience with setbacks and mitigation techniques through the course of the project.

5.4 Anticipated Risks

Risk: Our sensor array won’t provide enough data points to allow for accurate prediction analytics to be performed. As our weakest area of exposure is data analytics there’s no one on our team that has a good enough grasp on the scope of this portion of the project to rule this out.

Mitigation: Our design to use a series of microcontrollers to control and feed data from individual sensors leaves room for expansion. We are not currently using nearly every input port available to the controllers, and could add sensors to each system very easily. If the need for inputs ever exceeds our microcontroller we can simply scale up to a larger microcontroller.

Validity: This risk was never an issue for us, though our mitigating strategy was a good design decision, as for little cost it allowed us further expandability within our product.

Risk: Our team has no experience with Dart or AWS which are cornerstones to the software side of our project. There lies a risk that this inexperience may be insurmountable and hinder progress and push back deadlines.

Mitigation: We have fallbacks that we are more familiar with for both Flutter and AWS if the learning curve starts to hinder our progress in a meaningful fashion. The ece department offers VM and database services that we have utilized for previous classes, and could use to replace AWS if need be, four team members have experience with these systems. Three team members also have experience coding directly for android using java and Android Studio if Flutter proves to be a choke point. Both of these replacement opportunities are highly viable and still are free offerings, meaning using these as fail-state fallbacks would only incur costs of time.

Validity: This risk proved to be a valid concern and presented itself multiple times during the project. Almost immediately we decided to scrap Dart in favor of Flutter for the mobile app. This decision was made quickly and effectively in a manner that didn't affect the development process. The fact that we identified this as a potential risk helped make it clear to us when to abandon that technology for more familiar ground and it paid dividends toward productivity.

However, the same could not be said for our AWS troubles. The risk did present itself, but less abruptly than our mobile inexperience did. Small issues presented themselves throughout initial development, but were quickly reportedly overcome. It was however, very late in the design process that we realized our inexperience with the platform had led us to make several nonoptimal decisions about the structure of our backend. At this point we were too far along in development to in good faith switch platforms as our mitigation plan suggested, which led to a reconfigure a large portion of the project without delaying the rest of the timeline.

5.5 Experienced Risks

Risk: Availability and workload fluctuations meant that some timelines quickly became infeasible or extraneous.

Mitigation: Open communication played a large role in mitigating as much of this risk as possible. While strict adherence to timelines is ideal, we recognize that our group members are human and that this project was not always at the forefront of everyone's priority list. Through flexibility and understanding we were able to continue working, jump into gap areas, and continue moving the project forward.

We no doubt experienced many other risks during the design process but these form a large representation of the types of risks we experienced: uncertainty in design, unfamiliarity in tools, and unpredictable outside influences.

5.6 Lessons Learned

We learned many things over the course of this project. Some of those things were technical skills ranging from circuit design, application design, proper testing procedures, and familiarization with popular development platforms like Amazon AWS. Additionally we learned many non-technical skills such as task management in group development, how to approach development obstacles, as well as areas of improvement when it comes to process design and project management.

6 Conclusions

We leave this product in a functional, yet unfinished state. That is, we recognize there is so much untapped potential beyond what our final design delivers, that while out of scope for us may greatly enhance the product and its marketability. This section contains some closing thoughts on where development could continue from here.

6.1 Closing Remarks

While our final product matched our initial design requirements, we close finding that it is still in many ways incomplete. That however isn’t inherently negative, but rather opens up a world of possibility for further development and potential beyond what we could foresee at the time of initial development. The prospect for expanded functionality has us very excited, and as such we’ve compiled a list of future developments we feel could greatly increase the effectiveness or scope of our project.

6.2 Future Enhancements

Context Switching: The core of this project was occupancy sensor nodes that can deliver prediction modeling based off of human presence. While we took a focused approach with an application tied directly to improving process flow within a restaurant environment, we believe strongly that our project could be used in many other contexts as well. Example contexts that would require slight amounts of redesign are movie theater seats (detect absent moviegoers or presence in unsold seats), sporting stadiums, meeting rooms, or even classrooms.

These events are all primarily seated situations would could benefit from accurate occupancy sensor readings. The primary innovations that need to be done to adapt our project to these situations are redesign of the front end application and analytics algorithms in order to accurately reflect information that is desirable for the new use case. New sensors may be necessary for some of these situations, but as previously discussed our microcontroller approach purposely leaves room for expansion or deletion in the sensor array.

Integration to Queuing System: Currently our design helps align users with the waiting times at restaurants that have chosen to implement our structure. It does not however, allow a user to place themselves in a queue to be seated at such a restaurant. Doing so would require a massive upgrade for the system, as well as need to require certain safeguards as to not allow inflation of perceived waiting times. This would likely be a much larger undertaking than it seems, but is the most direct improvement to our specific use-case.

Algorithm Upgrades: Our knowledge about data analytics was the most lacking and stressed resource during the development cycle. While we are happy with what we've accomplished we are not naive enough to believe that our analytics algorithms are by any means close to perfect. The design could continually be improved and there may even be some machine learning opportunities to tie in if desired, but we maintain that by our design process any further upgrades to such analytics should easily slot into our existing backend structure, primarily affecting data between two datastores.

Security: As we put an ever increasing number of items on the internet we open an ever-expanding list of targets for adversaries. Given enough time it would be beneficial to extensively ramp up the security around our devices (making minimal info available from the devices themselves), ensuring proper encryption protocols are taking place between all stages of transportation (TLS 1.2+, etc), as well as organize an external pen-test against our systems, and do validation of the findings. Were this something we were to pursue as a marketable project this would be our top priority.

6.3 List of References

1. M. Wu, T. J. Lu, F. Y. Ling, J. Sun, H. Y. Du, "Research on the architecture of Internet of Things," Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, vol. 5, pp. V5-484-V5-487, 20-22 Aug. 2010.

2. K. Shinde, P. Bhagat, "Industrial process monitoring using IoT", I-SMAC (IoT in Social Mobile Analytics and Cloud) (I-SMAC) 2017 International Conference on, pp. 38-42, 2017.

3. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in the Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 93- 104, 2002.

4. P. Mudge "Self-powered long-life occupancy sensors and sensor circuits" U.S. Patent US6850159B1, 2005