Lab 1 – Pest Patrol Product Description

Anton Rasmussen

Old Dominion University

CS 411

Professor James Brunelle

October 12, 2022

Final Version

1 Introduction

Pest encounters happen frequently and occur widely across the United States but, due to their distributed nature, are a problem that appears to occur randomly (US Census Bureau, 2021). Neighborhood specific *Facebook* pages and the *Nextdoor* Application page for communities both contain more than a few incidents related to pests or encounters with pests (i.e., frequent and widely occurring encounters); but, because this reporting is ad-hoc and not centralized to a "pest-specific" application, much of the encounter data gets lost in the high volume of otherwise unrelated distributed communication (i.e., seemingly random encounters). Pest encounters happen but the true frequency of these encounters is all but lost because there is no single source to compile pest encounter data specifically.

Even in cases where pests are entirely preventable, because of the lack of information regarding where pest "hot spots" are within a community, avoiding an encounter is akin to contending with and trying to manage completely random probabilities. For example, the knowledge a family in a community gain when it overcomes infestation is knowledge that is unable to easily flow to others because there is no good, central, place to share lessons-learned, and therefore the family's lessons learned does not aid in the ability for others in the community to take proactive measures. Additionally, one finds that because there is no central repository of pest information for a certain community or region, planning of outdoor activities and events comes with risks that could otherwise be mitigated more easily were there some "place" to host this information.

The Pest Patrol application is designed to address the issue with aggregating pest-specific information into one easy to use application so that a community can coordinate the fight against pests. Pest Patrol plays the role of "do one thing well" in that its design is based on filling the gaps that exist in the currently distributed ad-hoc pest reporting space. While aggregating the information allows for more transparency around the ongoing “State of the Union” for pests in a certain area, the primary benefit of this application is making it possible for communities to pro-actively take measures to respond to pests, therefore taking back some determinism in an otherwise seemingly stochastic problem space.

Pest Patrol is not just an application for aggregating pest information related to a specific area but is itself tailored to act as a “community” in the sense that users are able to share about their encounters, interact with each other to obtain helpful and often crucial key details surrounding pests, as well as trigger insight into where pests may be coming from/moving to, where they might be more or less prevalent, and/or how intense the population of pests is for a given area of concern.

Ultimately the community aspect of Pest Patrol is the key driver of keeping individuals informed, minimizing the risk of pest encounters based on user reporting, projecting future pest movements/patterns, and providing an assist in and avenue for risk-mitigation when planning outdoor activities that otherwise may have resulted in a spoiled outing, a minor injury, or worse.

2 Pest Patrol Description

Pest Patrol is a cross-platform web application that relies on community-based (crowdsourced) information aggregation tied back to location services that help to map pest-related information geographically within the community. The network effect of this community-based crowdsourcing affords the ability to raise near real-time awareness through the application’s reporting mechanism because as more people use the application, more areas are monitored for pests, and, ultimately, more people fall into those “monitored” areas, thus driving the incentive for even more people to use the app. As each report is associated with a location (a location that can be made to be more- or less-granular based on the user’s choice), users themselves become both the source of pest encounter information as well as the consumers of related pest encounter reporting. The ultimate power of this location information is that the intensity of pest encounters can be mapped (e.g., using a heat-map) and used for predictive modeling of pest movements over an area (e.g., based on season, pest-type, etc.).

As users of Pest Patrol work through the process of addressing any kind of pest problems directly affecting them, they can share the knowledge they learn to other members of the community with the benefit of ultimately slowing or stopping the spread of pests throughout the community. As more and more data are collected related to pests and pest encounters, it will be possible to provide help in how to prevent pests as well as how to stay safe should one find themselves in a pest encounter.

The two main goals of Pest Patrol are 1) to make communities safer by broadening pest awareness in a community and thereby lessen encounters 2) to minimize the damage that specific pests can do to an area by aggregating information to help to slow pest spread and predict future infestations.

2.1 Key Product Features and Capabilities

The key features of Pest Patrol begin with the fact that the application is web-based and, therefore, multi-platform. Opting to implement a web-based application instead of an Android or iOS native application means that the application will remain easy to use no matter what type of device a person will be using. Given that a major factor of Pest Patrol is the community crowdsourcing aspect, it is important that the ability to use the app is not hampered by the technology.

Setting up an account with Pest Patrol is not a pre-requisite to benefit from the app’s data. However, without being a registered user, access is limited to read-only. Again, this design choice tips its hat to the idea that Pest Patrol is meant to be community-based and thus easy for all community members to use (indeed that’s the key-differentiator of this app versus similar apps on the market); however, because there is a desire to maintain a healthy community, any user who has access to write data to the application for others to see (e.g., a pest report, user comments, etc.) will have to be validated and their credentials verified.

The main screens a user will interact with are the Dashboard, a landing-page interface that affords the ability to navigate to anywhere else on the site, which provides three separate modes: *Incident Map*, *Discussions*, and a *Hybrid* experience where the Incident Map and Discussions are all on the same screen. The Incident Map is where reported incidents populate and includes attributes such as location, incident timestamp, pest type, submitter, report incident, and heat map. All these attributes are self-explanatory but what’s notable is that the attributes are customizable, filterable, and will adjust based on one’s location settings. In addition to user-activated functions there are also community discussions that populate based on discussion threads as well as pest alerts that send notifications of pest encounters. The discussion and private messaging features of Pest Patrol give it the true community power that is necessary for such an application to thrive in the marketplace. That said, the alerting and filtering features of Pest Patrol provide the community with the true power of achieving the goals of making communities safer and minimizing the damage of pests.

2.2 Major Components (Hardware/Software)

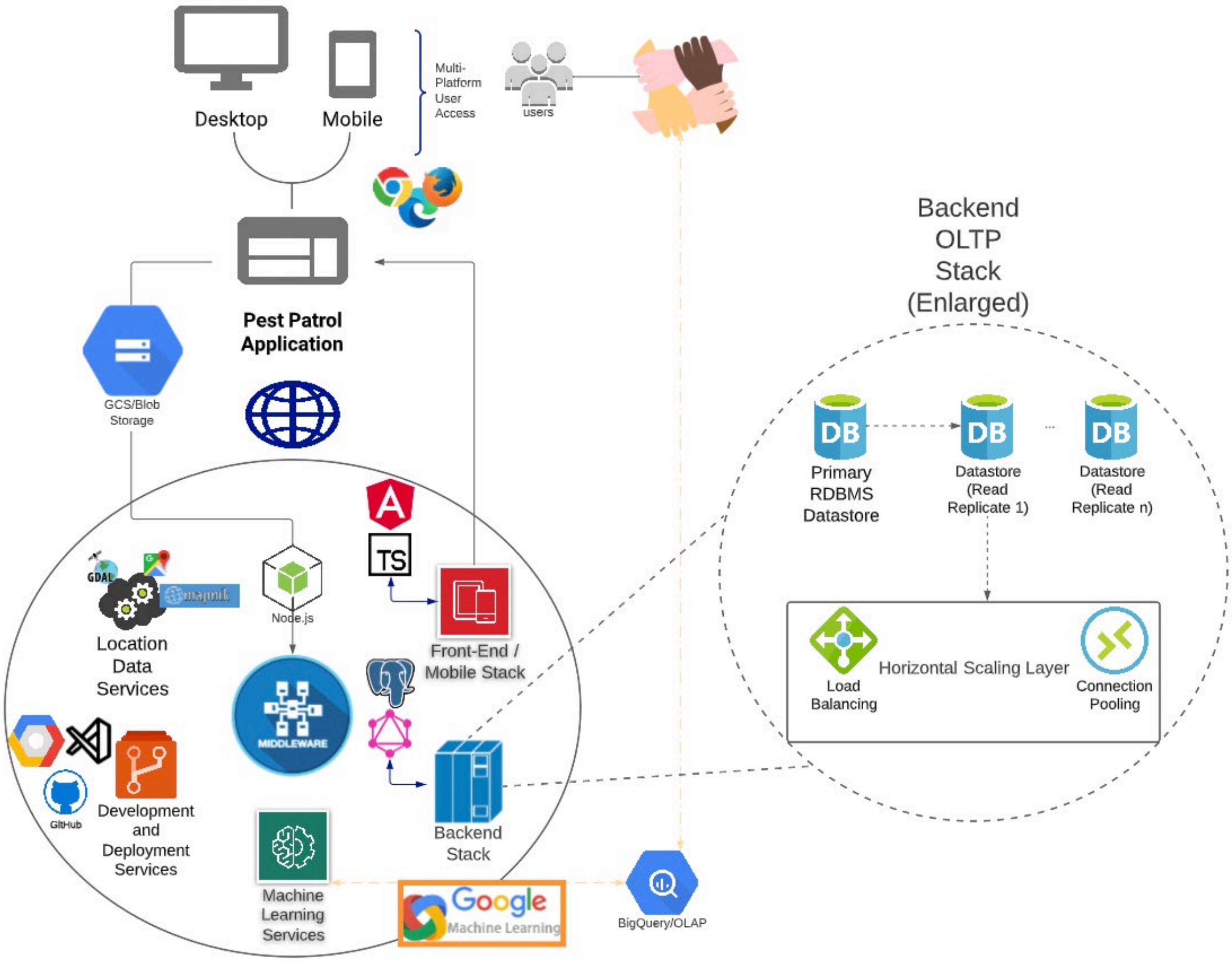
The overall architecture design of Pest Patrol is that of a multi-platform Web Application (Web App). As a Web App, Pest Patrol will allow easy access so long as a user has an Internet enabled device. Access to a camera and a touch screen device or other peripherals (e.g., keyboard and mouse) are all that is needed from a hardware standpoint. The major hardware and software components and their relationships are illustrated in Figure 1: Major Functional Components Diagram.

Because Pest Patrol will be hosted in the Google Cloud Platform (GCP), the need for servers and other “infrastructure” hardware has been abstracted away from the user. The key software component for Pest Patrol is a modern web browser able to handle responsive JavaScript (note that the Pest Patrol Web App will be tested against various device sizes using the browser’s devleopment tools and against the latest versions of at least three different browsers (Chrome, Firefox, and Edge)). The Web App’s primary software can be broken up into *front*-end and *back*-end stacks, tied together with middleware: in order to create a responsive JavaScript experience Pest Patrol will use a framework such as Angular.js and TypeScript (a type-safe version of JavaScript, which Angular uses) in the front-end; in order to write the APIs (REST endpoints), which serve up data from the backend into the UI, the prototype will use Node.js middleware; the back-end will consist of a PostgresSQL Managed SQL server via Google Cloud SQL. The front-end and back-end services for Pest Patrol will run in Google App Engine.

Lastly, though rather opinionated, the code-editor of choice for developers working on Pest Patrol, given the nice UI and the power of awesome plugins, will be Microsoft Visual Studio Code; that said, Code-editing is a personal matter so this will not be mandatory. We *will*, however, firmly commit to the use of a GitHub Repository and GitLab CI/CD integrations that will work through GitHub commit triggers as this tool set is widely used in industry and at ODU.

Figure 1

Major Functional Components Diagram (MFCD)



One can begin looking at the whole MFCD from the users' point of view. Given use of a responsive front-end stack, users have multi-platform access to the Pest Patrol application through either mobile or desktop devices. Between the user accessing the application through the front-end / mobile stack and the data stored in the back-end, there exists a layer of middleware services. In the middleware one ideally has location data services (that allow for viewing and tagging geo-location of pests), the typical front-end/back-end messaging found in web applications, and CI/CD services (that allow the dev team to quickly address application fixes and improvements).

3 Identification of Case Study

Pest Patrol is targeted at community members, hikers/campers, outdoor businesses, and cities. While the main user of Pest Patrol will likely be that same market of individuals dispersed across several apps reporting their encounters with pests (so-called "community members"), there are also potentially significant opportunities for targeting toward people who spend a lot of time outdoors, thereby becoming veritable "super" users of the app. Therefore, the first Case Study will be a hybrid study of both community members and outdoor enthusiasts. The expansion to outdoor businesses and cities will minimize downside risk while maximizing upside potential if Pest Patrol can target the application toward both community members and those who are frequently outdoors. For example, one of the major pests that people would be very interested in being alerted to are ticks (given the risks that these pests carry disease). In certain places tick populations are so prevalent that it is almost impossible to go on a short walk in the woods and not come home with a few crawling on one's clothes. In these cases, it would be both helpful and useful for hikers, campers, and other outdoor enthusiasts to have an easy way to log their encounters, including their exact location of encounter and any pictures they may have that help to identify the species of tick with which they have come in contact. In addition to being contributors of this data, other hikers and campers would be informed of the locations with the most densely distributed tick populations and thereby know to avoid those areas during their hike or when looking for a place to set up camp. Given the value of this information it makes sense that businesses oriented toward outdoor activities would also want to both contribute to as well as consume information from Pest Patrol. Ultimately, by providing the app freely to community members as well as hikers and campers through targeted ad campaigns via social media (gaining users with the least informational friction) and in-person demonstrations in outdoor equipment retailers (leveraging the business relationships naturally following from the utility of Pest Patrol's service), the market research gained from this first case study will enable the shipment of a more effective product to market in a much more timely manner.

In addition to outdoor enthusiasts and the companies that serve them, there is a market for cities, pest control companies, HOAs, government agencies, and academic researchers to be contributing members of the Pest Patrol community. Based on information gathered from the case study of Pest Patrol adoption by community members and hikers/campers, there should be ample data to help transition targeted advertising toward these larger markets.

4 Pest Patrol Prototype Description

4.1 Prototype Architecture (Hardware/Software)

Pest Patrol is a cross-platform web application that relies on community-based (crowdsourced) information aggregation tied back to location services that help to map pest-related information geographically within the community. The prototype of the Pest Patrol application will be a proof of concept that serves to demonstrate key elements of what will be the final product. The major hardware and software components and their relationships of the prototype are illustrated in Figure 2: Prototype Major Functional Components Diagram.

The prototype application has three main software stacks: the front-end, the back-end, and the middleware. We explore in detail each of these stacks below:

Front-End

In its ideal state, the prototype's frontend stack will consist of an Angular web application running on Google App Engine within the Google Cloud Platform. That said, because of resource and time constraints the minimum viable state of the protype's frontend stack consists of an Angular web application running in Docker containers. The Front-End stack consists of an Angular framework built in a Docker container with the latest Node.js image from the Docker Central Repository. This front-end image holds the Angular binaries and any node modules necessary for the application to run (e.g., the Google Maps libraries, Angular Materials library, etc.)

Middleware

Like the frontend, an ideal state for the prototype's middleware involves Node.js running on Google App Engine within the Google Cloud Platform. Again, however, the minimum viable state of the prototype's middleware consists of Node.js running in its own Docker container. The middleware Docker image consists of an Alpine version of Linux operating as a Node.js and Express server, able to send and receive HTTP requests between the front-end Docker container and database Docker container by way of Docker Compose.

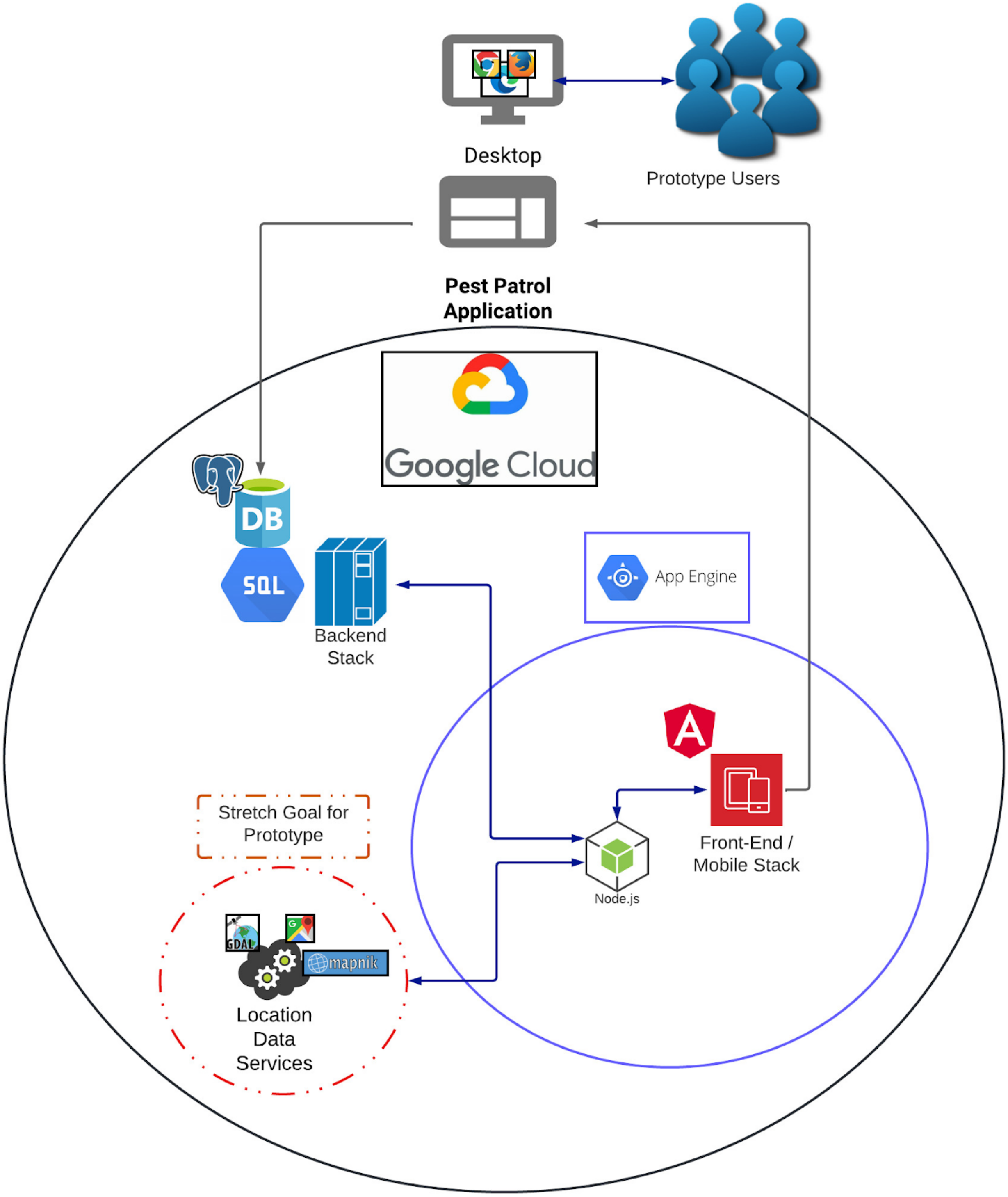
Back-End

Just like the frontend and middleware, the ideal state for the prototype's backend involves a PostgreSQL instance of Google Cloud SQL managed service persisting all the data for the prototype in a lossless way. However, the minimum viable state of the prototype PostgreSQL instance will, like the frontend and middleware, run in its own Docker container. Additionally, because a Docker container does not naturally lend itself to persistent data storage, it will be necessary to include an import step in Docker Compose for ingesting CSV files to load data each time the Docker container is initialized. Any writes to the database will persist for as long as the prototype Database is running in the same container (i.e., each time the container image is re-created all data will be refreshed with what is in the CSV files.

This Space Intentionally Left Blank

Figure 2

Prototype Major Functional Components Diagram (MFCD)



4.2 Prototype Features and Capabilities

The prototype seeks to employ the most important features and capabilities from the product design phase while eliminating features and capabilities that serve as enhancements or that are not primary to demonstrating a proof of concept of the Pest Patrol Application. All features and capability differences between the Real-World Product and the Prototype are compared in Table 1: *Pest Patrol Real-World Product vs. Prototype Function List*; however, in the next few paragraphs each stack's major capabilities are outlined:

Front-End

The frontend prototype allows us to provide a responsive web application with emphasis on a high-quality UI experience. With a responsive frontend it is possible to demonstrate Pest Patrol's key features and functionalities in both a desktop and mobile view via the browser. In addition to the responsive framework, the frontend makes use of several libraries to enhance the UI experience: Google Maps API, Angular Materials, Location Data Services.

Middleware

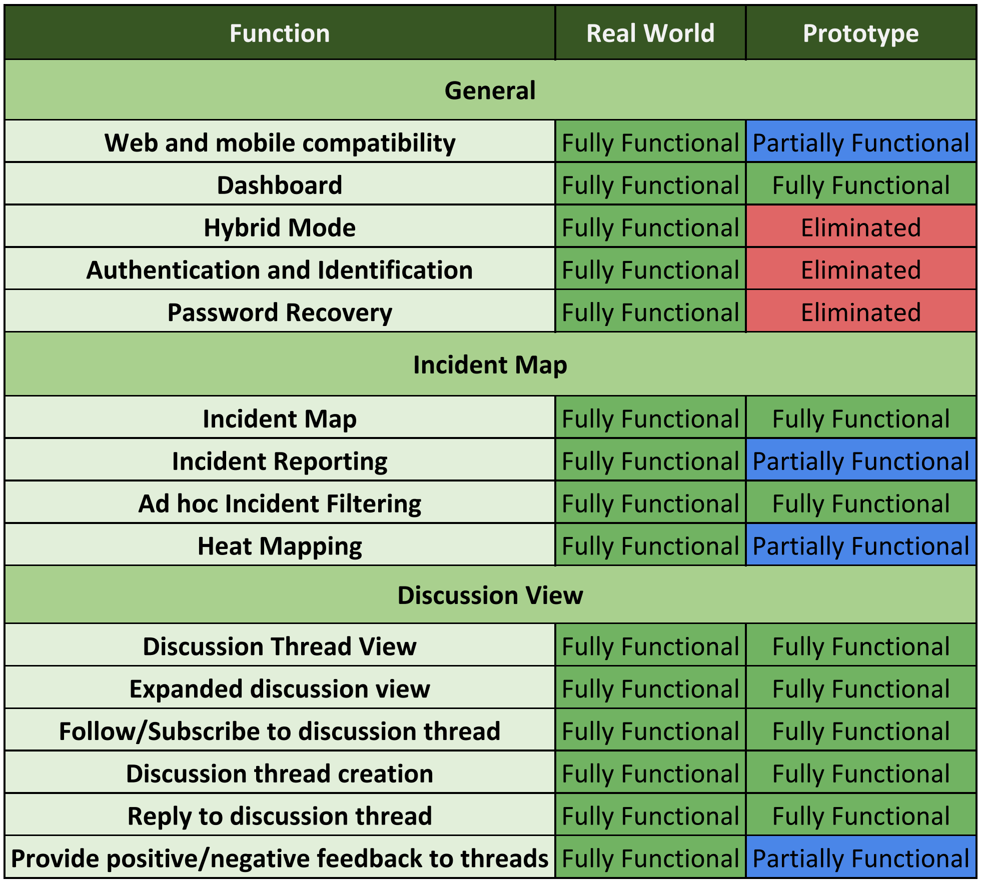
Node.js middleware holds the control flow logic to queries for each endpoint. Triggers for each query are based on requests sent from the frontend using POST requests (thus helping to prevent URI SQL injection). The middleware (ideally) sends prepared statements to the back end and (minimally) receives query responses in a JSON format, which it serves up to the frontend, thus demonstrating useful design and the employment of risk mitigation techniques.

Back-End

By leveraging PostgreSQL server, our backend employs the transactional query efficiency of a Relational Database Management System, which provides timely query results in all of the prototype's use-cases. An additional (ideal scenario) feature of having all data stored within Google Cloud SQL is that the security of our data is managed by Google. The initial prototype will have all images stored on disk and references to the image location will be stored within Google Cloud SQL for retrieval. However, as a stretch goal, it may be possible to explore storing image data in Google Cloud Storage, a blob storage "NoSQL" solution popular for similar use-cases.

Table 1

Pest Patrol Real-World Product vs. Prototype Function List



Function	Real World	Prototype
General		
Web and mobile compatibility	Fully Functional	Partially Functional
Dashboard	Fully Functional	Fully Functional
Hybrid Mode	Fully Functional	Eliminated
Authentication and Identification	Fully Functional	Eliminated
Password Recovery	Fully Functional	Eliminated
Incident Map		
Incident Map	Fully Functional	Fully Functional
Incident Reporting	Fully Functional	Partially Functional
Ad hoc Incident Filtering	Fully Functional	Fully Functional
Heat Mapping	Fully Functional	Partially Functional
Discussion View		
Discussion Thread View	Fully Functional	Fully Functional
Expanded discussion view	Fully Functional	Fully Functional
Follow/Subscribe to discussion thread	Fully Functional	Fully Functional
Discussion thread creation	Fully Functional	Fully Functional
Reply to discussion thread	Fully Functional	Fully Functional
Provide positive/negative feedback to threads	Fully Functional	Partially Functional

Pest Alerts		
Pest Alerts	Fully Functional	Partially Functional
Alert customization	Fully Functional	Partially Functional
Community		
Search for user	Fully Functional	Fully Functional
Add friends	Fully Functional	Eliminated
Report Users	Fully Functional	Eliminated
User reputation system	Fully Functional	Eliminated
Automated Moderation (ML)	Fully Functional	Eliminated
Hide flagged content	Fully Functional	Eliminated
Account suspension	Fully Functional	Eliminated
Flag inappropriate content	Fully Functional	Eliminated
Content removal	Fully Functional	Fully Functional
View flagged content	Fully Functional	Eliminated
Block user	Fully Functional	Eliminated
Content search	Fully Functional	Fully Functional
Recent Neighborhood Activity	Fully Functional	Fully Functional
Direct Messaging	Fully Functional	Eliminated
New thread activity notification	Fully Functional	Fully Functional
New direct message activity notification	Fully Functional	Eliminated
New incident notification	Fully Functional	Fully Functional
AI generated notifications (ML)	Fully Functional	Eliminated
Notification customization	Fully Functional	Partially Functional
Predictive Modeling (ML)	Fully Functional	Eliminated

4.3 Prototype Development Challenges

The main expected challenges to be encountered while completing the prototype are best divided between technical and non-technical challenges. The technical challenges can be further split by technology "stack."

For the front-end technology the main challenges are use of a complex web application framework (in our case Angular) and the ability to write code in type-safe JavaScript (Typescript); needed familiarity with the MVC architectural pattern, Style Sheets (e.g., CSS), and HTML; knowing how to develop the frontend in a way that does not get blocked by the back end (e.g., use of stubs or mock data); knowledge of error handling; and ability to write unit tests.

For the middleware technology the main challenge entails needing the ability to write APIs that connect our frontend UI to our database server. In order to successfully write these APIs developers will need to understand how asynchronous JavaScript works, how the HTTP request/response protocols function, and how SQL Prepared Statements function. Additionally developers will need to have an ability to write SQL Queries to return data in the correct way, an ability to work with JSON files, knowledge of error handling, and ability to write unit tests.

For the back-end technology the main challenges are the need to understand the best design practices for an RDBMS; ability to create schemas that are normalized properly; an understanding of query performance and how indexing works; knowledge of how to develop in the back-end in a way that does not block the front-end (e.g., creating temporary tables with mock data so that front-end developers can do their work while creating the final data structures as different tables).

The non-technical challenges in creating a useable prototype can be sub-divided into two categories: development challenges (the known unknowns) and unforeseen challenges (the unknown unknowns). The main challenges to development are developer experience, working as a distributed team, the large scope of the project (both in terms of coding and non-coding tasks), working locally vs. on the Cloud, and the front-to-back-end integration pipeline. The main unforeseen challenges could be: cloud and location services costs and deployment complexity.

5 Glossary

Administrator: Individuals responsible for keeping an application running. For Pest Patrol these responsibilities include the moderation of user interactions and posted content.

Angular: A Typescript based open-source web development framework.

Bot Moderation: The automatic screening of user content to ensure proper user behavior.

Community Member: A member of a community, see Community definition.

Community: The people with common interests living in a particular area broadly the area itself.

CRUD: Create, Read, Update, and Delete. They are the four basic operations of persistent data storage.

Density: The number of incidents linked to a certain pest in a given area.

Discussion Thread: Running commentary of messages between members within a community

Docker: an application that uses operating system level virtualization to deliver software in packages called containers.

Feedback: A positive or negative rating that can be applied to any user-created discussion thread or response.

Geo-tagging: The process of appending geographic coordinates based on the location of a mobile device.

Geo-targeting: Method of determining the geolocation of an application user and delivering different content to that visitor based on their location.

Git: A distributed version control system for software development.

GitHub: An internet hosting service for software development and version control using Git.

Google App Engine: A cloud computing platform for the development and hosting of web applications in Google-managed data centers.

Google Location Services: A service from Google that aims to provide a more accurate device location and generally improve location accuracy.

Heat Map: A data visualization technique that shows the magnitude of a phenomenon as a color in two dimensions.

Hiker/Camper: Any individuals that engage in prolonged leisure activities outdoor.

Homeowner's Association (HOA): A membership organization formed by a real estate developer to own and maintain common green areas, streets, and sidewalks and to enforce covenants to preserve the appearance of the development. It is operated for the benefit of all the residents of the community.

Incident Map: A graphical map that displays the locations of all reported pest incidents for an area.

Incident: An occurrence or sighting of a pest reported by a user.

Infestation: A specially designated incident involving a high concentration of pests in a given area.

Instance: A single copy of the software running on a single physical or virtual server.

Interface: Front-end graphical displays of the Pest Patrol application that users interact with.

Major Functional Component Diagram: A high-level visualization of the main system resources and their dependencies with one another.

Node.js: an open-source, cross-platform, back-end JavaScript runtime environment that runs on a JavaScript Engine and executes JavaScript code outside a web browser.

Pest: Any animal or plant harmful to humans or human concerns.

Pest Control Company: Any business entity that specializes in the regulation or management of pest species.

Pest Hotspot: A number of reported incidents of a specific pest that exceed a certain threshold determined by the user.

PostgreSQL: An open-source relational database management system emphasizing extensibility and SQL compliance.

Predictive Modeling: The use of statistics to predict outcomes.

SMS Messaging: Short Message Service. Is a text messaging service that uses standard communication protocols to enable the exchange of short text messages.

TypeScript: Open-source programming language that is a syntactical superset of JavaScript.

User: Any individual interacting with the Pest Patrol application. This includes: community members, hikers/campers, pest control companies, homeowner’s associations and administrators.

Virtual Machine: A compute resource that uses software instead of a physical computer to run programs and deploy apps.

VSCode: a source-code editor made by Microsoft that includes features for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

6 References

Abell, J. (n.d.). Nuisance Wildlife Encounters on the Rise. https://www.homestead.org/lifestyle/homesteading-life/nuisance-wildlife-encounters-on-the-rise/

Bugg, S., Colborne, S., & Haerther, D. P. (2020, March 2020). Great Lakes Invasives: Sea Lampreys. https://www.sheddaquarium.org/stories/great-lakes-invasives-sea-lampreys

Coyle, D. (2021, September 10). Not So Fast! International Biosecurity Program Succeeds in Preventing Spread of Invasive Moth. https://entomologytoday.org/2021/09/10/international-biosecurity-program-succeeds-preventing-spread-invasive-moth-lymantria-dispar-asiatica/

Herring, D. (2012, March 6). Climate Change: Global Temperature Projections. https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature-projections

How Warmer Winters Affect Pests. (2018, January 15). https://varmentguard.com/blog/warmer-winters-affect-pests

Japanese Beetle Repeatedly Eradicated from California. (n.d.). https://www2.ipm.ucanr.edu/Invasive-and-Exotic-Pests/Japanese-Beetle/

Lyme Disease Costs Up to $1.3 Billion Per Year to Treat, Study Finds. (2015, February 5). https://publichealth.jhu.edu/2015/lyme-disease-costs-more-than-one-billion-dollars-per-year-to-treat-study-finds

Parkman, K. (2021, September 23). Pest control statistics and trends. https://www.consumeraffairs.com/homeowners/pest-control-statistics.html

U.S. Census Bureau. (2021, April 21). Residents of 14 million housing units reported seeing roaches, 14.8 million saw rodents in last 12 months. https://www.census.gov/library/stories/2021/04/how-many-american-homes-have-pests.html