San Diego Zoo Wildlife Alliance - Sensor Dashboard

BRANDON DALPORTO, AHMED HUSSAINI, JIPING LIN, MALCOLM MCSWAIN, AKSHAYA SUNDARAM, and ETHAN TAN, UC San Diego

Abstract

The San Diego Zoo Wildlife Association (SDZWA) manages a vast network of multimedia sensors spread across thousands of acres. These sensors are deployed and managed by researchers with hopes of collecting various data formats, from numerical readings, weather station reports, video feeds, and audio feeds. The current infrastructure employed by the SDZWA requires physical connection and hands-on monitoring for sensor status - this implementation is not practical for data collection or scalable. In order to more efficiently manage these nodes, our group proposes a web-based application equipped with a user interface to view sensor output, metadata, and health - all managed via a secure, reliable database capable of collection and storage of sensor information. For this, we have implemented a full stack application utilizing React and Node.js with a SQLite database to build a "home-base" dashboard that will centralize access to these sensors. This web application will allow researchers to easily view and manage data across large networks, resulting in more accessible information and higher potential for meaningful scientific results.

CCS Concepts: • **Software and its engineering → Software creation and management**; • **Human-centered computing** → *Interaction design*; *Visualization systems and tools*; • **Applied computing**;

Additional Key Words and Phrases: dashboard, sensors, embedded systems, UI/UX design

1 INTRODUCTION

1.1 Background

The San Diego Zoo Wildlife Alliance is a global nonprofit conservation organization that researches animal behavior in order to deepen scientific understanding of many different species. In their mission statement, the SDZWA makes it clear that they aim to develop sustainable conservation solutions through wildlife care, education, and research. [20] The San Diego Zoo Wildlife Alliance manages not only the San Diego Zoo and Safari Park, but various research sites as well. All of these locations encompass thousands of acres of land, containing both small and open-range enclosures housing countless animal species.

In collaboration with the University of California San Diego's Engineers for Exploration, we have partnered with Ian Ingram at the SDZWA in order to create a solution for managing sensors located within the research domain of the San Diego Zoo's grounds [19].

1.2 Motivation

Researchers at SDZWA need to manage a variety of sensors in the animal enclosures that continuously collect multimedia data formats, including video, audio, and numerical. For example, animal enclosures may contain a live video feed to monitor behavior, a microphone to accompany it, and a variety of environmental data monitors like temperature and light sensors. These sensors provide the organization with critical data on animal health, habits, and safety.

The present need at the SDZWA is sensor data collection as well as sensor health monitoring. Currently, there is no centralized location that collects, stores, or visualizes this data. Furthermore, there is a need to track sensor status and metadata to know when sensors go offline or have connection problems. Our team's goal is to build a full stack web application that serves all these needs with a clean, responsive, and intuitive user interface.

Sensors at the SDZWA may be geographically distant from each other and lacking internet connectivity therefore requiring a sort of "home-base" to communicate with, indicating proper functionality and allowing the transfer of data collected. From our understanding, there is currently no infrastructure in place to support such operations, and any sensor managed by the SDZWA must be physically interacted with for management

purposes. This model simply does not scale, and requires more time spent by researchers on checking sensors, rather than the research topic at hand. We propose our web-application solution as a method of solving this data problem and allowing for researchers at the SDZWA to focus on the information from the sensors rather than the sensors themselves.

1.3 Proposed Solution

Research is paramount to the conservation and protection of endangered species and wildlife in general, and sensors are an integral part of the SDZWA's research process. There is no central manner in managing these sensors, so research and studying of these animals can become a tedious process.

In order to efficiently manage access and data to the SDZWA's network of animal research sensors, we propose a web-based user interface to aggregate sensors' output and metadata. Our web application is implemented with a React front-end connected to a Node-based API that's integrated with a SQLite database.

2 RELATED WORKS

2.1 Dashboard Applications

Dashboard applications are typically used for managing information about products or services. They serve as centralized visualization and consolidation of large amounts of data. They may contain user functions such as making requests, editing, or adding information, as well as graphs,charts, or other metrics of visualization given a data stream. Other common components of dashboards include a navigation bar to centralize the different operations a user may perform, as well as a landing screen to display various types of information cleanly and efficiently. Common applications of dashboard applications are home monitoring systems, such as Home Assistant, which provides users with important information based on sensors and other IoT devices placed within their home [2]. Home Assist may be a helpful tool at home, but does not meet the standards of data integrity required by scientific research. There are various manners in which dashboards can display information, one of them being a tiling system.

2.2 Tiles

Tiles, or cards, are components of application frameworks that consolidate information into easily-readable and eye-catching sections that may be arranged vertically or horizontally. This is a commonly used philosophy in web applications and software, with a notable example being the Windows 8 start screen from Microsoft [4]. This start screen can be considered a dashboard, giving users the ability to navigate to any application they wish at a quick glance. While visually appealing, another important aspect of tiles are their flexibility. We aimed for our Sensor Dashboard to be completely responsive, with a mobile first approach. Tiles provide a suitable manner for displaying info on both wide and narrow screens because of the modularity that allows us to create a flexible grid system in React-Bootstrap.

2.3 Object Relational Mapping

Object relational mapping, or ORM for short, is a way for developers to interact with a database using objects. This layering helps simplify database interactions by helping developers spend less time developing SQL code, and more time writing application code to interact with the database. An added benefit of ORMs is their security against SQL injection attacks which can occur when user queries are applied directly to database code when an ORM is not used. In this project, ORMs are being used to simplify the database interactions to help manage our REST API for sensors as well as the administrative back-end, coming with the added benefit of being an extra security layer against SQL injection attacks. [15] [18]

2.4 SQLite Database

SQLite is a library that implements a serverless, self-contained, zero-configuration SQL database engine [3]. This project utilizes SQLite to manage the database, primarily because of it's lightweight features, that is it does not require a server or an operating system for it to be functional and is fully integrated in our application. The lightweight and serverless properties of SQLite allow for a mobile database that may be hosted on any device, potentially in a remote location; hence, SQLite is the best candidate for the sensor communication and management network.

2.5 Docker

Docker is an open source containerization platform which enables developers to package applications into a container (with application's components, dependencies, and necessary OS) so that it can be run in any environment, and it provides interfaces to simply and safely create and control containers [5]. In creating our application, we use Docker for hosting purposes, notably the Node.js server, as well as containerize our application so that it can be easily configured and run on various platforms and architectures as required by the SDZWA.

2.6 Synthesis

With all of these technologies and applications in mind, our group has designed and developed a full stack web application that utilizes the design principles involved with a dashboard view and use of tiles, supported using React as an interactive and responsive front-end user experience, with data being stored on a lightweight SQLite database (with Prisma as its ORM), and Docker as the pedestal to hold everything up and allow for cross architecture support and mobility.

3 PROCESS & METHODOLOGY

In terms of project organization, our group found it paramount that development be sectioned into two sides: a front-end and back-end team in order to reduce team-wide clutter regarding resource sharing and project management [6]. After sub dividing members into their respective teams, documentation was created by both sides about project goals in regards to what the front-end as well as back-end MVP would look like, and how they would integrate together to create a complete SDZWA sensor dashboard. In this manner, a clear picture of what could be realistically achieved by the end of the quarter was formulated, from which both teams could orient themselves in terms of their own progress timelines. Using an Agile-like framework [12], milestones were generated with various sub-goals, each of which an individual team member would be responsible for managing and ensuring its completion. Due to time restrictions that come with the 10-week quarter system, major milestones were on a (approx) two week schedule, with 3 minor milestones being completed within that time frame as well. Teams met in weekly sprint meetings to update each other on progress and in the final weeks, focus was placed on integration of the front-end and back-end between both teams. The integration process was a collaborative effort between the two teams, with common milestones and shared responsibilities.
Methodology in creating the sensor dashboard was primarily in the realm of making an application which was lightweight and easily deployable. The project methods can be boiled down to a few phases: research, design, prototyping, feedback, implementation, integration.

3.0.1 Research. The initial phase of the project was research, a necessary step in ensuring that we create a product that meets industry standards, and is capable of meeting all the needs of the SDZWA. The primary goal of our research phase was to determine what our application stack would look like, and what the SDZWA could support. For example, when deciding what type of database we wanted to use, there were many options out there, such as mySQL or SQLite. After doing a literature survey and speaking internally, we decided that SQLite would be the best choice for our application, as it is lightweight and portable, potentially allowing for use on a

Raspberry Pi (or the like) and placed in a location with physical access to the sensors. From the research phase, we composed a technology stack of React [16], Node.js [13], SQLite [17], Prisma [14], Docker [8], as well as design ideas based upon popular web applications.

3.0.2 Design. As with any user facing application, design played a big part in developing our product. With goals of usability, modularity, extendability, and security in mind, we created a thorough design process for implementing this system. front-end design considerations mainly consisted of responsiveness and usability - for this, we created a UI/UX mock-up for the proposed user experience, and designed our React framework for a responsive page system. On the back-end side, extendability and security played large roles in database design. The database had to be extendable in the case of new sensor types, as well as secure in handling user and sensor data. To meet these requirements, schemas were created and verified prior to the prototyping phase.

3.0.3 Prototyping. The prototyping phase consisted of creating a bare-bones application that had some features, and simulated those missing from our designed system architecture.[7] To do this, the front-end team created a basic React web application, which consisted of all the proposed pages with simulated components (tiles, video player, forms, etc.). While this application had no functional use, it served as a proof of concept for our initial design that ensured everything looked and felt right from the user perspective, as well as served as a means for outside feedback.

3.0.4 Feedback. In seeking feedback on the initial prototypes, our group presented mock-ups and the basic React app to Chris Crutchfield (UCSD), Nathan Hui (UCSD), and Ian Ingram (SDZWA). This step provided valuable criticism and suggestions for user experience as well as implementation details [11]. From this, our group gathered what works, what does not work, and what is still needed to meet all of the requirements at hand.

3.0.5 Implementation. In order to meet milestones for the MVP, implementation tasks were divvied up and individuals were responsible for the oversight of tasks, with support from other members if needed. Many of these individualized tasks were carried out in isolation, with code review sessions afterwards to ensure correct implementation and functionality. For larger milestones where more than one person was responsible, pair / group programming sessions would occur, following an implementation plan to ensure that code was written with goals in mind, and to minimize bugs. Testing was an ongoing process in the implementation cycle, with different inputs being generated and passed into front-end and back-end applications, searching for potential issues that might arise from a general user's experience.

3.0.6 Integration. Integration was the last and most crucial phase of this product's development, as the front-end and back-end team had worked relatively separately over the quarter. Spearheaded by front and back-end leads, constant communication of requirements and progress was upheld, with group programming and debugging sessions being held to ensure a smooth integration of applications.

3.1 MVP

All of the development stages listed above had the goal to meet a common Minimum Viable Product (MVP) by quarter's end. While a complete product was the end goal of this project, requirements were divvied up based on common, front-end, and back-end requirements.

3.1.1 Common. The web application must support offline use, as well as HTTPS communication for security purposes.

Both of these requirements have been met, as any React or Node.js application may run locally on a device, and be accessed via IP in a LAN (Local Area Network). One potential implementation using this would be having the hosting computer along with all the sensors connected to a network switch and handle all traffic locally,

with no outside connection to the internet. HTTPS is also not a problem, as there is native support in React and Node.js, allowing for encrypted communication between components in the system.

3.1.2 Front-End. The front-end built using React and Node has several capabilities as part of the minimum viable product. The dashboard must pull data from multiple sources via URL, and handle various types of data, including numerical, video and audio. The application also requires user authentication in order to access the dashboard, and a user must be invited prior to registration. We will not be integrating with the zoo for this project, so mock data pulled from the web is used to display template sensors.

The front-end team has met the quarter defined MVP, aside from the user invitation process through the implementation of our React web application. The current iteration of our application pulls data from a sample set hosted within the front-end as a component, as well as via URLs. These dashboard tiles are capable of displaying numerical data via components, and video + audio data through react-players. Authentication to the dashboard is handled via tokenized log ins, and forced redirect to the registration page if a user attempts to access the dashboard without a token. The token is acquired from the back-end at log in or registration time and is valid for 2 hours. Mock data used as a part of the MVP proof of concept involves numerical data from the visx library [1], sample mp4 audio, and video linked via URL. Due to time restrictions and logistics, the front-end did not implement the user invitation process. Primarily because user invitation is not a task for the front-end, and especially not a React app, we did not find this an appropriate milestone for our team, and plan to work with the back-end to implement this in the future.

3.1.3 Back-End. The back-end built using Node.js, SQLite, and Prisma has three primary components of the MVP: the back-end must be capable of hosting an online database that may interact with the front end, built with an Object Relationship Manager (ORM), with API functions for reading, updating, creating, and deleting sensor and user information. The administrative back-end helps to manage user authentication using a REST API, with support for create, update, and delete sensor operations. Using Prisma, an Object Relationship Manager, the back-end is capable of creating, managing, and visualising tables contained in the database. Lastly, there are routes and controllers as a part of the back-end to define queries based on REST API operations (PUT, POST, GET, DELETE) implemented in Express, a Node.js framework. With all of this, the back-end MVP has successfully implemented an available and local datastore that may be deployed on any device using a Docker container, without connecting to the internet. In addition, the back-end has adapted a simple data model that supports the expected API functions of the front-end, and may be extended easily in future iterations of the data model.

3.2 Milestones

As with all organizational choices in this project, we have created a list of Milestones to keep track of and ensure constant and meaningful progress throughout the quarter. Milestones were organized as shared, front-end, and back-end. Front-end milestones revolved around creating the React web application. Back-end milestones were primarily concerned with database configuration, operations, and API functions. Shared milestones consisted almost completely of integration tasks, with class deliverables falling into this category as well. The following table (Table 1) does not encompass all tasks completed over the quarter, as there were many more class deliverables, and sub goals leading up to the major milestones.

3.3 Quarter Plan

As mentioned in Section 3.0, our group has created a detailed quarter plan in order to work effectively, and meet deadlines at an accelerated pace. Composed of 50+ GitHub issues, our quarter plan can be shown at a bird's eye view in Figure 1.

Week	Team Responsible	Member Responsible	Description
5	front-end	N/A	Initial "Hello World" Web App
5	front-end	Malcolm	Registration and Authentication Flow
5	back-end	Akshaya	Create database schema for sensors
5	back-end	Jiping	Create database schema for users
6	front-end	Ahmed	Dashboard Tile View
6-8	back-end	Ethan	Rest API Templates
7-9	front-end	Brandon	Graphing and Data Visualization Functionality
8	back-end	Ethan	Database Security
8-10	Both	Ethan + Malcolm	System integration
10	Both	Brandon + Akshaya	Final Report, Final Video

Table 1. Milestones

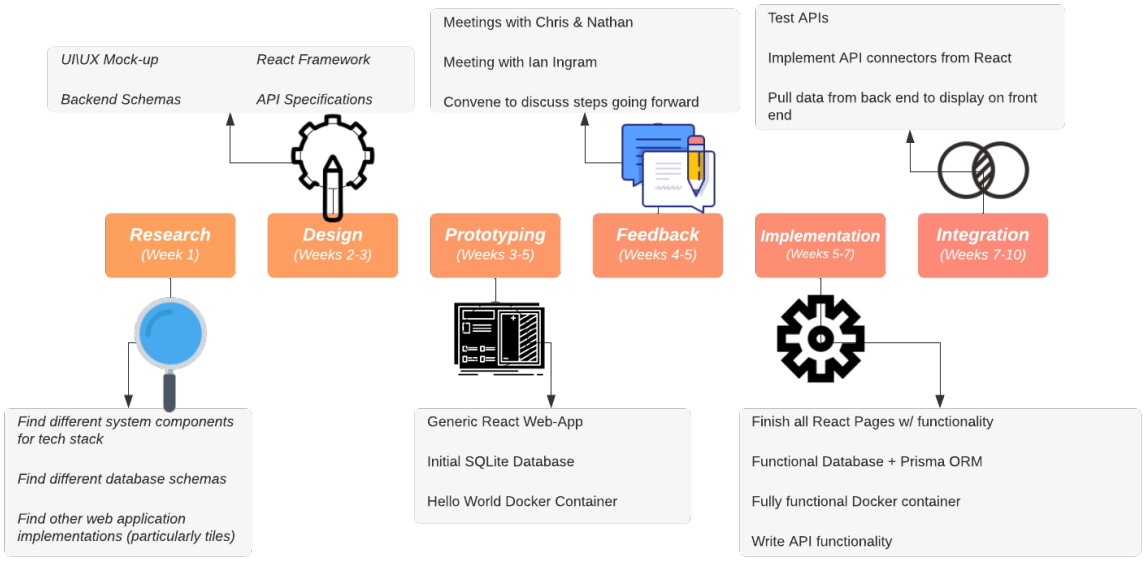

Fig. 1. Quarter Plan.

4 IMPLEMENTATION

The following sections will describe the front-end and back-end implementations of the React web app, along with the administrative back-end. See Figure 2 for a high level diagram of the system architecture.

4.0.1 Front-End. We leverage React to build the application using library components. and Next for server-side rendering. React-Bootstrap components are used to build the forms and look of the login, registration, edit sensor, sensor, and dashboard pages. We employ React inline styling to make the application responsive to screen size changes, so the dashboard can be easily used on desktop as well as mobile. React hooks are used to change the state of the components when a user performs an action that triggers an API call, such as hitting the "Log-in"

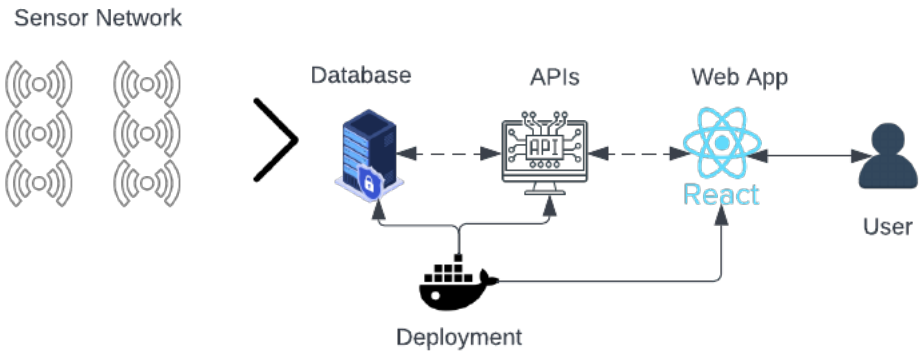

Fig. 2. System Architecture

button. We navigate between the Next pages using the router to "push" the page we want to the front. Integration with the back-end is done using the JavaScript fetch method wrapped in functions that make requests to the back-end server and receives responses back.
In terms of the page layout, the application launches to the dashboard, which will redirect you to the registration screen if a user is not signed in. Here users can fill in personal info to make a new account. From the registration screen, you can also navigate to the login screen by utilizing the navigation bar. Again, logging or signing in triggers an API call, and once you arrive to the central dashboard one can see the sensors laid out in tiles (if there are any), and an option to sign out at the top right of the nav bar. There is also a plus button that allows you to add your own sensor. Clicking this will navigate you towards the add/edit sensor page, where you input detailed info about the new or old sensor such as the type of data and URL it's pulling from. Completing this form and submitting will trigger another API call to the back-end. We can also visit specific sensor pages by clicking the tiles, which will give us more information on that sensor and also allow us to delete or edit the sensor. Deleting the sensor will trigger a REST API call in our front-end as well.
See Figure 3 for the communication flow that occurs between pages of the React app, note that API calls are sent to the back-end.

4.0.2 Back-End. In terms of back-end database implementation, the choice was made to use SQLite for the purpose of its server-less configuration which can run on many different platforms. This was particularly important because of the fact that sensors would be making use of these back-end systems and so the database which was utilized needed to be something that was able to support embedded software. Object Relational Mappers (ORM) were also utilized in order to simplify the the schema creation process. Using an ORM also comes with the added benefit of increased security against SQL injection attacks. With these two technologies as the base of the project, a simple REST API in Node.js was created for registering and editing sensors as well as users for the sensor dashboard. See Figure 4 for a high level component diagram. We store only the encrypted string of the admin password in the database, and its salt. For security of sensitive information like password, the direct string is not stored in the database. The choice was made to use Prisma as the ORM of choice due to its ease of implementation alongside SQLite. Table 2 and table 3 show the details of our database schema for User and Sensor tables:

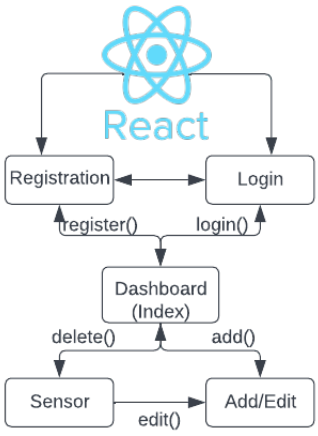

Fig. 3. front-end Communication Flow

Beside designing the databases, we also created a Docker file for deploying our schema using Prisma on SQLite back-end using Node.js server on any device. We also created REST API templates for accessing, and managing the information in the database.

The REST API templates for accessing sensor table are:

`getAllSensors, getSensorById, getSensorByName, updateSensorById, deleteSensorById, createSensorByName`

The REST API functions written for accessing the user table in the database are:

`newUser, LoginUser, changePassword, getUsers, getUserById, getUserByEmail, updateUserByEmail, deleteUserByEmail`

. The overall communication flow between the different tools, we used in the back-end is shown in Figure 4.

5 RESULTS

The culmination of an academic quarter's worth of work is a fully functional sensor dashboard web application. The front-end product provides a seamless user experience with responsive pages and data visualization. The dashboard view is capable of all CRUD operations involving sensors, and may communicate all necessary

Variable	Data Type	Description
user_id	Int	Primary ID to identify the admin user of the application
firstName	String	First Name of the admin user
lastName	String	Last Name of the admin user
email	String	Email of the admin user (unique)
password	String	Encrypted string of the password (the password string is not directly stored)
salt	String	salted hash of the password that is used for encryption

Table 2. User Table

Variable	Data Type	Description
id	Int	Sensor Primary ID used for referencing the sensor within the database
name	String	Name of the sensor that the zoo uses to reference it (Unique)
description	String	Description or Note that describes the sensor and its function in the zoo
type	Int	Integer referring to the type of the sensor (e.g. Camera, Motion, Thermal)
source	Int	Integer referring to the source of the sensor (e.g. Numeric, Audio, Video)
locationX	Float	Latitudinal Location of the sensor in the zoo
locationY	Float	Longitudinal Location of the sensor in the zoo
status	Int	Integer referring to the status of the sensor (e.g. Idle, Active, Not Working)
installedAt	Float	The date and time when the sensor was installed at the zoo
updatedAt	Float	The data and time the sensor was last updated
url	String	The storage location of the data collected by the sensors

Table 3. Sensor Table

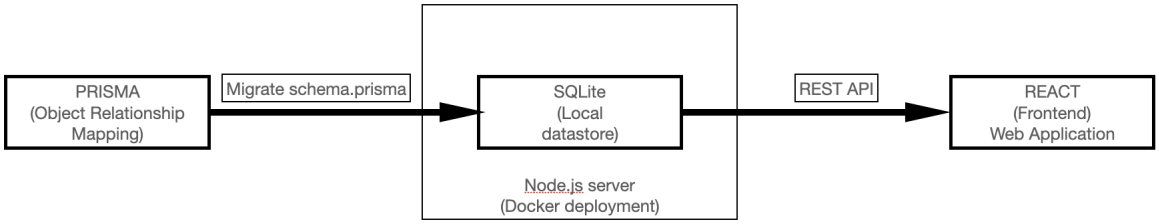

Fig. 4. back-end Communication Flow

information to the back-end using REST API calls. The administrative back-end is equipped with user and sensor data schemas to store and make use of various objects. All user information is stored in a secure manner, making use of salted and hashed passwords. Equipped with REST API functionality, as mentioned above, the back-end is capable of receiving and sending all information required to support user and sensor data within the web application.

All code from this project may be found on UCSD's E4E GitHub [9] [10]

6 FUTURE WORK

Due to time constraints of the quarter, this is not a completely finished project, and there remains work to be done.

6.0.1 Front-end. Currently, our dashboard is utilizing sample data along with capabilities for an audio-player and video-player using different React player libraries. The SDZWA's audio and video files may be of the RSTP stream format and would be live, so there is quite some work to be done to implement RSTP capabilities in the dashboard. We also did not gain access to any of the zoo's sensors, so integrating production sensors into the front-end would be a huge step forward in development. From the front-end perspective, there remains work to be done with regard to adding more types of visualizations. We are currently using the visx library for our sample data, but there is room to add any other type of data visualization to this project. We simply import these visualizations as a component and place these in the dashboard, so adding new ones would be as simple as adding a new component. There also remains work that was planned out for implementing a sensor health graph that would be placed on

the individual sensor page. This graph may be implemented as a bar chart, with an hourly/daily/weekly report on whether the sensor is communicating/partially communicating/not communicating. To implement this chart, one may pull a bar chart from the visx or another library and transfer data from the back-end into that chart. Currently, there is no back-end support for this data, but that may be added very easily. There will always be work to be done from a design and user experience perspective, but there are currently no plans for any changes in these regards.

6.0.2 Back-end. In terms of future work which is to be done on the back-end, 6the data models/schemas can be extended to fit more metadata information about the many different kinds of sensors used in the SDZWA. Due to the nature of the constraints which were given to us, our team was only able to make a single mock schema for a theoretical sensor which is not specialized. By adding more data models which are more distinct to information collected by specific sensors, more information could then be displayed by the dashboard, and thus a better overall view of sensors could be gained by dashboard users. Secondly, more API functions could be used to create, update, read, and delete sensor and user information from the database. Our current selection of functions is adequate, however it could never hurt to include additional functionality from the front-end side in terms of how to manipulate users and sensors. An additional functionality may be to add an extra layer of user groups above users in order to simplify how sensor access is managed in terms of what is displayed on a dashboard per user (and user group). Thirdly, more security features can be extended to ensure further data integrity in the database. Currently what is being employed is a simple salted hash of passwords to prevent passwords from being stored in plain-text in the database. Additionally, use of Object Relation Mapping helps to fight against SQL injection, however our team finds that additional measures may be put in place through implementation of SSL or site encryption to further ensure the security of dashboard and user information. Lastly, continuing on the back-end team hopes to create a more manageable abstraction layer which allows for the easy transfer of information from individual sensors to a sensor back-end of sorts. Currently, the working version of this dashboard is primarily focused on the administrative back-end and so managing to interface sensors with our application would finally make this a fully functioning project. There is added difficulties in this step however, due to the complexities of hardware requirements and the question of how sensors will be connected to any sort of layer which helps it interact with databases, thus making for the most pressing issue which must be resolved in terms of future work to be accomplished for the back-end portion of this project.

7 CONCLUSION

The complete software development lifecycle of this project has proved to be a rigorous task, simulating a bringing-to-market experience of industry. As a part of CSE 145/237D at UCSD, the goal of this course was to make a product at an accelerated pace, involving embedded systems. The first couple weeks of the course involved choosing a group, based upon already in place projects, or newly proposed ones. Our group had originally been partnered with the Institutes of Americas in order to create an eco-tourism web application. Up until week 4-5 of 10 in the quarter, this was the goal, with initial design choices being decided upon and hours of work involved. At week 4, the group was informed that we need to pivot to a new project, the SDZWA sensor dashboard. With 4 weeks shaven off, the accelerated nature of the production process was exacerbated. Through proper mentorship and project management, minimum requirements, goals, milestones, and tasks were rapidly developed and executed. Through incredible amounts of work and long hours, this group was able to create an MVP in approximately 5 weeks. From this, we have gathered that with proper organization and strong work ethics, software development can and does occur rapidly in response to the ever changing needs of a growing technology market. Upon completion of this academic quarter, our group has created a web application that can help researchers view and manage a tile dashboard displaying audio, video, and numerical data from various sensors in the San Diego Zoo Wildlife Alliance.

REFERENCES

[1] AirBnB. 2022. *visx Library*. https://airbnb.io/visx/areas

[2] Home Assistant. 2022. *Dashboards*. https://www.home-assistant.io/dashboards/

[3] ST Bhosale, Miss Tejaswini Patil, and Miss Pooja Patil. 2015. Sqlite: Light database system. *Int. J. Comput. Sci. Mob. Comput* 44, 4 (2015), 882–885.

[4] Zac Bowden. 2019. A look at the evolution of the Windows 8 Start screen. *Windows Central* (2019).

[5] Thanh Bui. 2015. Analysis of Docker Security. *CoRR* abs/1501.02967 (2015). arXiv:1501.02967 http://arxiv.org/abs/1501.02967

[6] A. Cockburn. 2000. Selecting a project's methodology. *IEEE Software* 17, 4 (2000), 64–71. https://doi.org/10.1109/52.854070

[7] Gabriele Montelisciani Gualtiero Fantoni Daniele Mazzei, Giacomo Baldi. 2018. A full stack for quick prototyping of IoT solutions. *Annals of Telecommunications* 73 (2018), 439–449.

[8] Docker. 2022. *Docker takes away repetitive, mundane configuration tasks and is used throughout the development lifecycle for fast, easy and portable application development – desktop and cloud.* https://www.docker.com/

[9] UCSD E4E. 2022. sdzwa-sensor-dashboard-backend. https://github.com/UCSD-E4E/sdzwa-sensor-dashboard-backend.

[10] UCSD E4E. 2022. sdzwa-sensor-dashboard-frontend. https://github.com/UCSD-E4E/sdzwa-sensor-dashboard-frontend.

[11] Margret Hazen. 1985. Instructional Software Design Principles. *Educational Technology* 25, 11 (1985), 18–23. http://www.jstor.org/stable/44424513

[12] Jim Highsmith. 2009. *Agile project management: creating innovative products*. Pearson education.

[13] Node. 2022. *Node.js is a JavaScript runtime built on Chrome's V8 JavaScript Engine.* https://nodejs.org/en/

[14] Prisma. 2022. *Next-generation Node.js and TypeScript ORM.* https://www.prisma.io/

[15] Prisma. 2022. Prisma Studio. https://github.com/prisma/studio.

[16] React. 2022. *React: A JavaScript Library for building user interfaces.* https://reactjs.org/

[17] SQLite. 2022. *SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.* https://www.sqlite.org/index.html

[18] ERIC CABREL TIOGO. 2022. *Understand the shadow database feature of Prisma ORM.* https://blog.tericcabrel.com/understand-the-shadow-database-feature-prisma/

[19] San Diego Zoo and Wildlife Alliance. 2022. *Ian Ingram, M.S. - Senior Researcher.* https://science.sandiegozoo.org/staff/ian-ingram-ms

[20] San Diego Zoo and Wildlife Alliance. 2022. *San Diego Zoo and Wildlife Alliance Mission Purpose.* https://science.sandiegozoo.org/who-we-are/mission-purpose