Advanced Animal Simulator: User Interface Design

A Design Project Report

Presented to the School of Electrical and Computer Engineering of

Cornell University in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by:	**Peihan Gao (NetID: pg477)**
	Fan Lu (NetID: fl427)
	Siyu Liu (NetID: sl3282)
MENG Field Advisor:	**Joseph Skovira**
Outside Field Advisor:	**Dan Fletcher**
Degree Date:	**January 2020**

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: **Advanced Animal Simulator: User Interface Design**

Author: **Peihan Gao (NetID: pg477)**

Fan Lu (NetID: fl427)

Siyu Liu (NetID: sl3282)

Abstract:

The project is a re-accomplishment according to the work delivered by the previous team. The newly programmed user interface retains all previous functions. Furthermore, compared with the previous work, the new user interface has a more modern appearance, better maintainability, and extensibility. The module-based structures and highly integrated interfaces make the entire project convenient to maintain and to extend in the future. The web-based front-end was constructed with React. The back-end server and database were constructed using Node.js, Express, and MongoDB. Additionally, the entire project has been successfully deployed and tested on Heroku.

Catalog

1. Introduction

This project is a re-accomplishment of the robot dog user interface (UI) according to an existing web-based UI. The UI delivered by the previous team was a PHP-based web page, which provides complete functions but appearance remains to be improved. Thus, designing a modern appearance is one of the basic focuses with top priority in this project. Furthermore, the scalability (multi-user performance) and maintainability are also substantially improved. In terms of the architecture, this project is reconstructed based on a classic three-tier structure (see Fig.1). The front end visually presents the user interface and reacts to operations from users. In detail, React is used for the logic implementation of the front end while Bootstrap and Material-UI are applied to the appearance designs. The communication between the frontend and backend utilizes HTTP requests when real-time data is required or forms are submitted by users. The backend works as the nerve center, which takes charge of the real-time data transmission and scenario running. The backend also manages a database (MongoDB), which stores scenarios, real-time vitals, hardware modes, and account information. Furthermore, integrated interfaces are also provided by the backend to interact with the hardware of the robot dog. This report intends to elaborate on the design methodology in this project. The corresponding discussion about the deliverables and issues for future maintenance will also be expounded on.

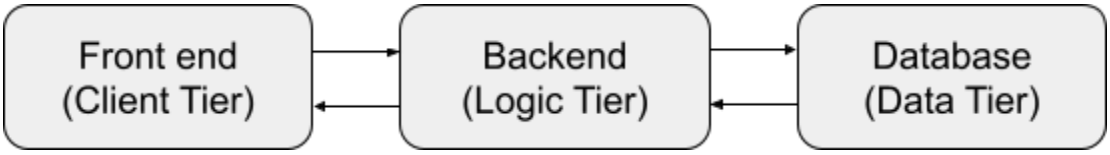


Fig.1 User Interface Structure

1.1 React

React is currently one of the mainstream JavaScript frameworks for constructing user interfaces. In detail, the declarative grammar and component-based development are the core strengths of React [1]. In terms of the components of React, well-structured components that manage their own state are built, encapsulated, and composed to make complex UIs. Components of React can be classified as either function components or class components. These function-based or class-based components can be imported and reused in the development of the parent components. Thus, React components present excellent development efficiency, code utilization, and maintainability.

1.2 MongoDB

MongoDB, which is a document-based database, can be classified as the NoSQL database [2]. In MongoDB, records are presented as the form of a document. In terms of the data format, MongoDB defines a style similar to JSON. Every document in MongoDB consists of field and corresponding value pairs. Data MongoDB will finally present a tree-shaped storage structure, where both the array and document can be the tree node. In terms of the advantages, high-performance data persistence is one of the core strengths of MongoDB. In detail, I/O operations in MongoDB are substantially decreased by introducing embedded data models. Another strength of MongoDB is that faster queries are supported by the data index technique, which employs keys from embedded documents and arrays for performance improvement.

1.3 Cloud Computing Platform - Amazon Web Service vs Heroku

In the first semester, our team has successfully deployed this project on the cloud servers of AWS. However, the deployment was migrated to Heroku later for cost consideration during the development phase.

AWS is a powerful platform, on which users can freely configure the server as their demands. The server services on AWS, such as a simple EC2 instance, provide an intuitive environment to easily manipulate the resources. While configuring the server, users can choose the Amazon machine image, which is basically the server operating system, static storage, and memory storage, security group, and load balancer [3].

Heroku is a totally free cloud platform despite being less powerful than AWS. Most of the configurations are fixed and not editable but have been sufficient to a small to medium scale application. Compared with AWS, Heroku is free and also convenient. The applications in this project are small-sized and the corresponding configuration can be relatively static. Hence, Heroku is completely able to satisfy the demands for creating the production environment.

2. Methodology and Deliverables

The front end of the user interface was constructed with React. React-hook, a new feature imported in React 16.8, was frequently employed to program the basic components. The backend was constructed with Express. Additionally, the definition of interfaces in this project follows the RESTful styles for efficient development and maintenance.

2.1 Frontend

The front-end was constructed with the React. The modularized React components were encapsulated layer by layer (see Fig.2) and were finally presented as a single-page application (SPA) with a tree-shaped structure. There is a router in the App.js, which parses the URL and displays the corresponding pages (encapsulated as components) in a specific HTML label. Additionally, an error page will be returned when visitors attempt to get access to nonexistent URLs (see Fig.3). In terms of the advantage of such a component-based structure, assuming that

a new hardware device is implemented at the robot dog, the user interface can be efficiently extended with only several lines of code.

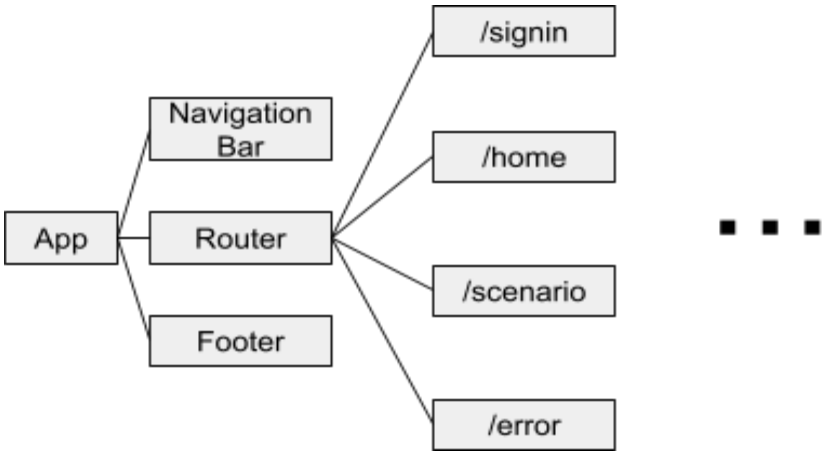


Fig.2 Front-end Tree-shaped Architecture

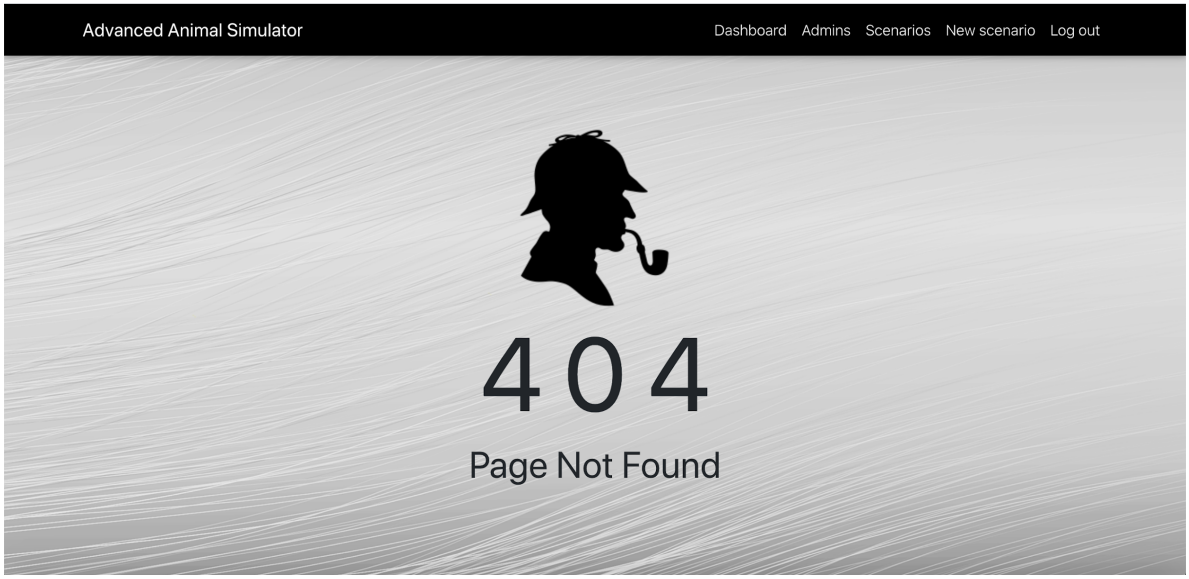


Fig.3 Error Page for Nonexistent URLs

User Authentication

With respect to user authentication (see Fig.4), both the frontend and backend will check the sign-in status of users. Any visitors without logging in will be redirected to this page if they try to enter the URL of the dashboard page. The logic function user authentication is programmed using React-hook and has been integrated into the file "auth-hook.js". The current logics will check the validation of the forms. During the process of signing up, email repetition will be checked, and existing email is prohibited. After receiving a pass token sent back from the backend, the login is completed. The detailed account information will be and the login status will be recorded for further front-end logic. The previous unaccessible URLs will also be unlocked in the router.

The sign-in/up page has been integrated into a single React component in the JavaScript file (auth.js), which can be directly imported or independently modified. In terms of the appearance, the sign-in/up button will be disabled and hints will appear when users submit an incomplete form or a form with incorrect formats. In the sign-up page, there is also an area for users to upload a profile image (see Fig.5), which will be shown at the top of the sidebar after the user logs in.

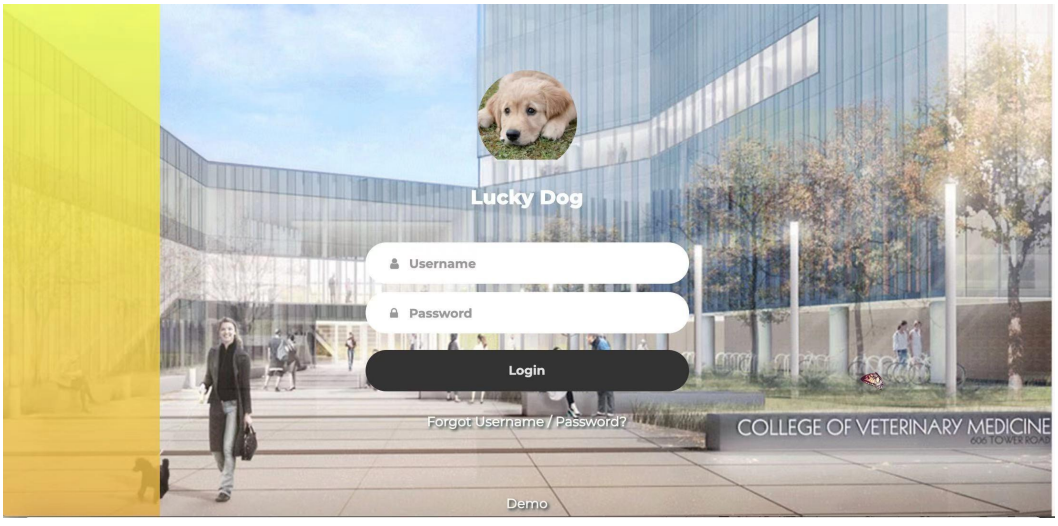


Fig.4 Sign-in/up Page

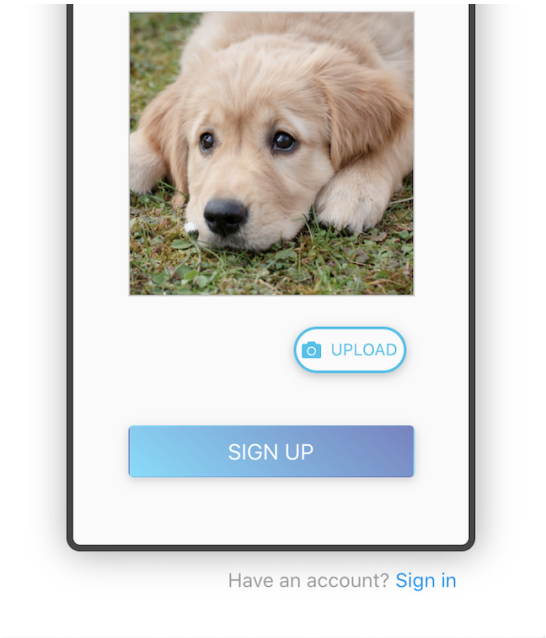
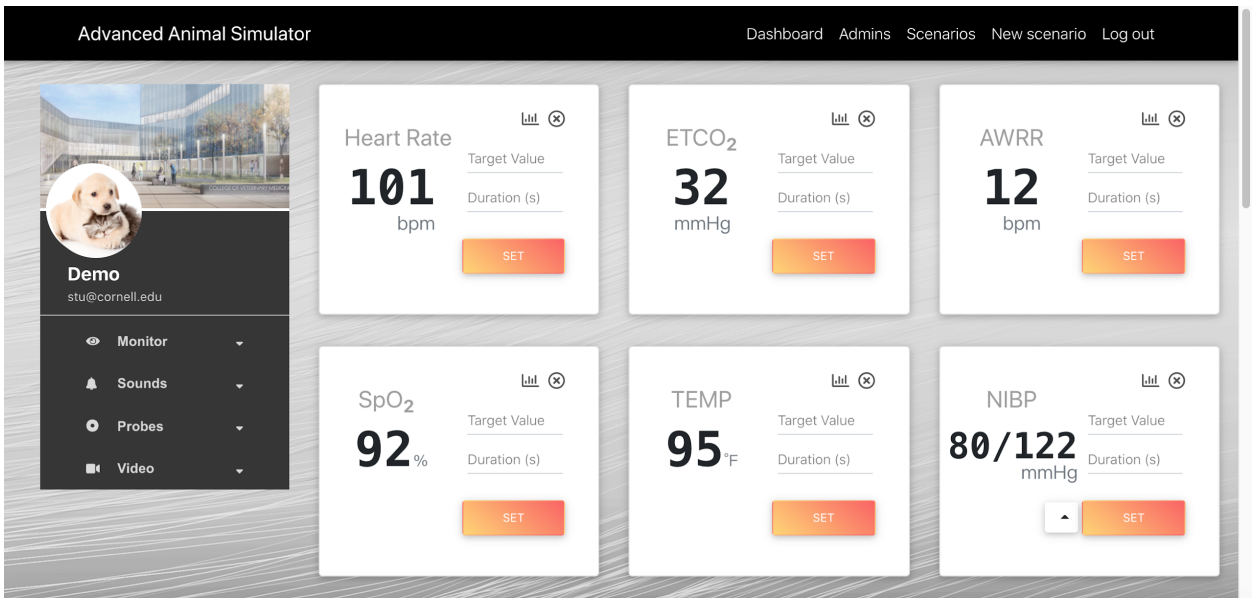


Fig.5 Image Uploader

Dashboard

The dashboard component consists of a sidebar and a card-based monitor (see Fig.6). The users can remove any card and add them back then as their wishes.



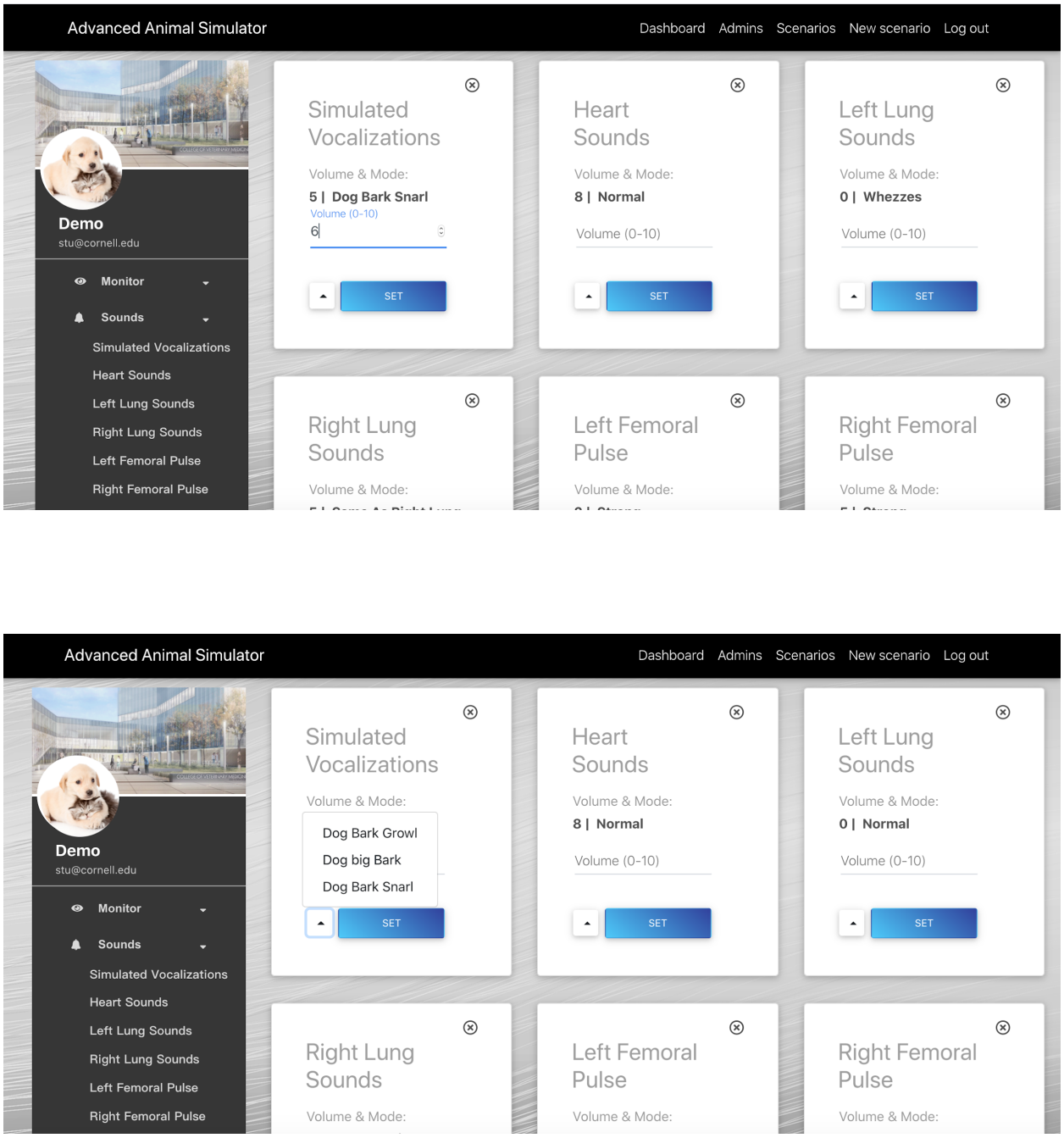


Fig.6 Appearance of Dashboard

The display of the dashboard is flexibly designed according to the Bootstrap grid rule. Thus, the layout of the webpage will be automatically rearranged when the width of the window is not enough (see Fig.7).

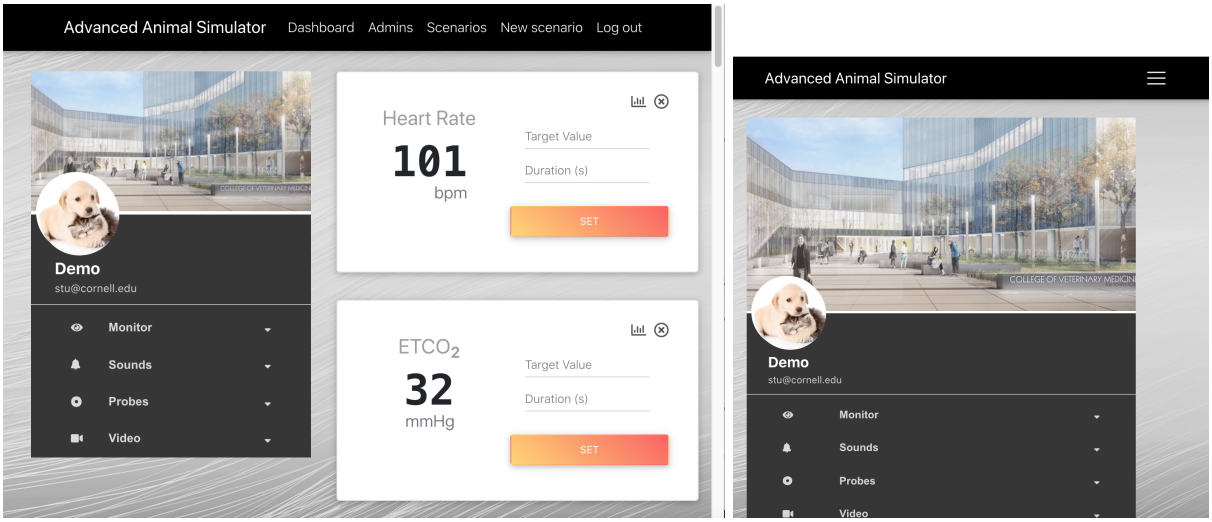


Fig.7 Appearance of Resized Dashboard

In terms of the card-based monitor of vital signs (with orange buttons) and hardware control panels (with blue buttons), the data and current mode are transported from the backend in real time. “Axios” is used as the third-party module to handle HTTP requests (The corresponding code is available under the path of “/src/request” in the project). As is shown in Fig.8 the data and mode will be refreshed every second using HTTP GET requests. When a form with both the target value and transfer duration has been submitted, an HTTP POST request will be sent to the backend (see Fig.9). After the frontend receives a response to the HTTP request, the status code in the response will be checked first. If an error occurs for some uncontrollable reason, such as bad network connection, the symbol of “--” will be displayed at the area of data (see Fig.10).

```
{data: {…}, status: 200, statusText: "OK", headers:
{…}, config: {…}, …}
  config: {url: "/api/vitals", method: "get", headers…
  data:
    values:
      probe: {chestMovement: true, cuffProbe: true, ec…
      soundpulse:
        heartSound: (2) [0, 8]
        leftDorsalPulse: (2) [4, 5]
        leftFemoralPulse: (2) [4, 2]
        leftLungSound: (2) [0, 0]
        rightDorsalPulse: (2) [1, 8]
        rightFemoralPulse: (2) [4, 5]
        rightLungSound: (2) [6, 5]
        simulatedVocalization: (2) [4, 5]
        __proto__: Object
      vital:
        awrr: 12
        diastolicNIBP: 80
        etco2: 32
        heartRate: 96
        nbpHR: 5
        spo2: 92
        systolicNIBP: 122
        temp: 95
```


Fig.8 HTTP GET Requests


```
{data: {…}, status: 201, statusText: "Created", header
s: {…}, config: {…}, …}
  config: {url: "/api/vitals", method: "post", data: …
  data:
    vital:
      createdAt: "2020-05-22T04:55:08.512Z"
      duration: 10
      previous: 101
      slope: -0.5
      target: 96
      __v: 0
      _id: "5ec75b2cd05a9040d4b4524a"
      __proto__: Object
    __proto__: Object
  headers: {content-length: "145", content-type: "app…
  request: XMLHttpRequest {readyState: 4, timeout: 50…
  status: 201
  statusText: "Created"
```


Fig.9 HTTP POST Requests

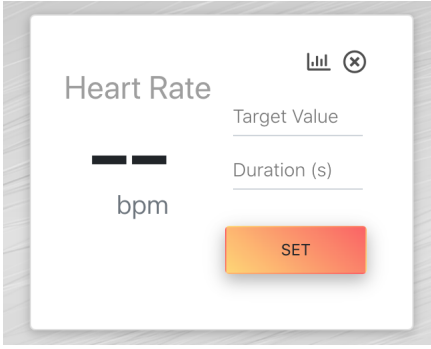


Fig.10 Network Unavailable

The historical record of each vital over the past five minutes (see Fig.11) can be viewed through clicking the chart-shaped icon in the top right corner of the corresponding card.

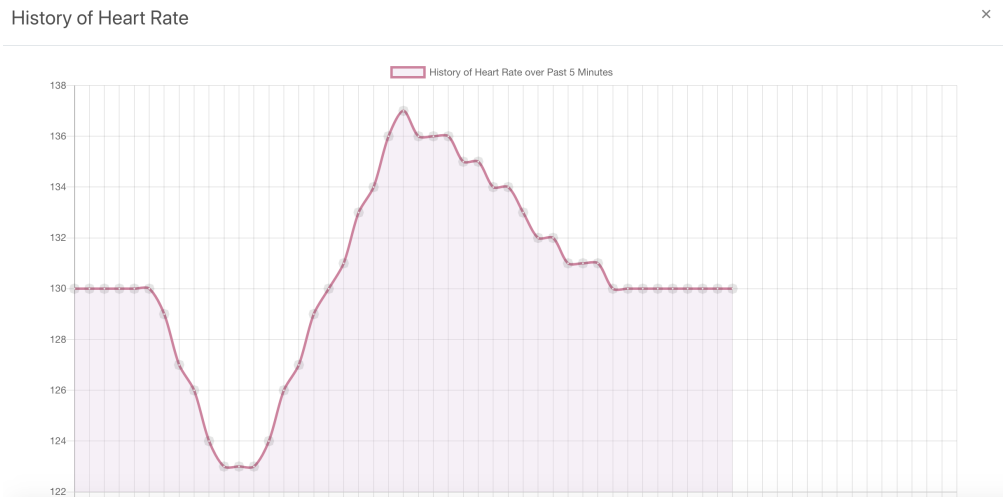


Fig.11 Chart for History Records

Scenario Selector

Users can enter the scenario selector through the navigation link in the navigation bar. Fig.12 shows the appearance of the scenario selector. After selecting a specific scenario, the user will be redirected to the dashboard page and the vital signs and modes will automatically change in the backend according to the pre-edited scenario.

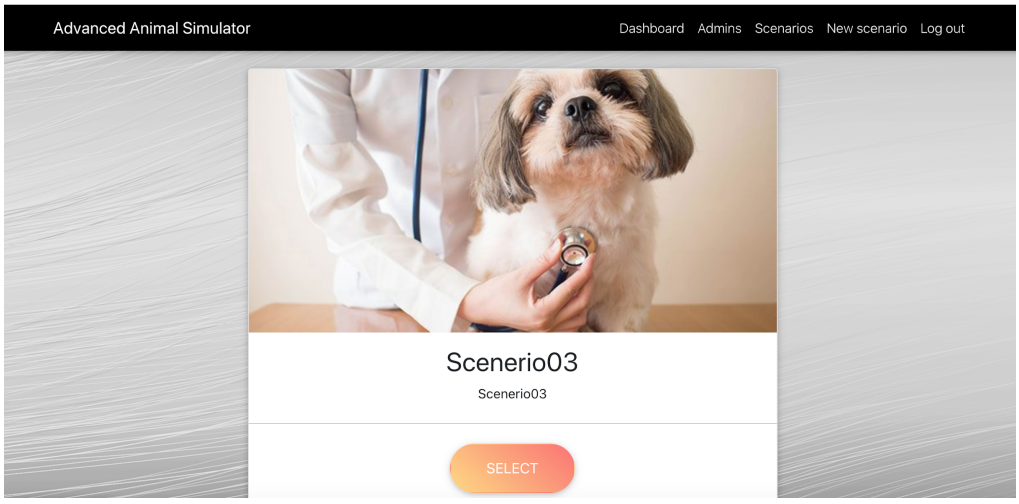


Fig.12 Chart for History Records

Video Player

Fig.13 presents a video player component for playing teaching videos.

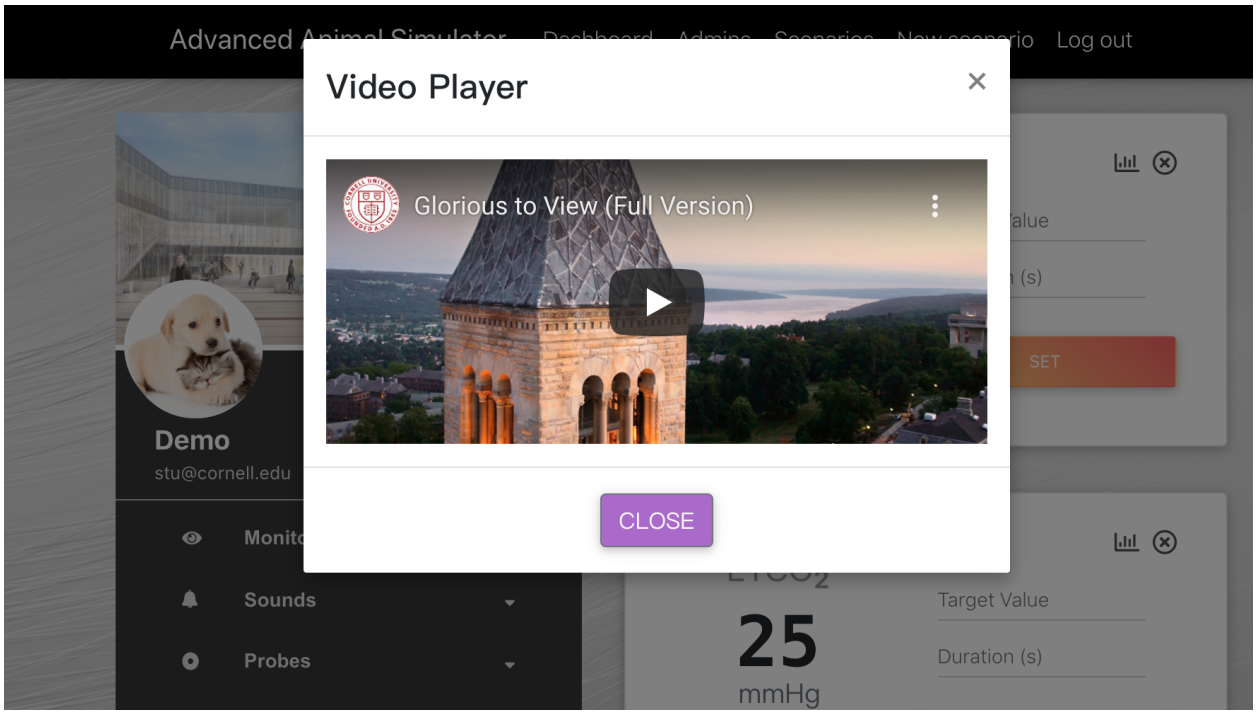


Fig.13 Chart for History Records

2.2 Backend and Database

User Authentication

The backend employs the JSON Web Token (JWT) for multi-user authentication and authorization. In detail, the main functions realized at the backend include automatic login, long-term login, login timeout, validation check, and repetition check.

With respect to the process of login, firstly, the backend reads account information from the database to check whether the user email exists. The value of "null" will be returned to a variable called "existingUser" when the email of the requester does not exist. For an existing account email, the user authentication will end after the password is verified. In the next step, the result

of the authentication will be sent back to the frontend using JWT, which is an open standard that defines a compact and self-contained way for securely transmitting information between the client and server as a JSON object [4]. In detail, “res.json(token: token)” is used to return the response with authentication results to the frontend. Finally, the web application in the front end will receive and confirm the token for further logics (For example, the previously unavailable URLs in the router will be unlocked after the user logs in).

Furthermore, the backend also supports the record operations of the “create”, “read”, “update” and “delete” (CRUD). In terms of the frontend, the web application only supports the approved users to create, read, update, and delete records. The majority of records associated with the user-id are stored in MongoDB while only the profile images of users are stored in the backend (The database only keeps the path of those images). Fig.14 shows part of the information of one particular user.

Fig.14 Chart for History Records

Real-time Data and Scenario

As has been previously mentioned, the frontend sends an HTTP GET request every second to refresh the data for vital signs and modes. After receiving an HTTP GET request, the backend will read from the database. Then all data will be encapsulated in the “promise” object and sent

back to the frontend. The modes, such as the strength of the left femoral pulse, are static and they can be simply transmitted. In terms of the vital signs, they can be changed within a specific transfer duration set by users. Thus, there is a program running in the backend to handle this process. Additionally, the scenario selector is also realized in this backend program.

2.3 Deployment

In the first semester, the entire project was deployed to AWS and ran as an Elastic Compute Cloud (EC2) instance. In detail, the Amazon machine image of the server was chosen to be Ubuntu, and default settings were employed for the load balancer, storage, and memory. Then the security groups for both HTTP and HTTPS protocols were added. Then the key-pairs were configured to enable SSH login. On the Ubuntu server, the Node.js, Express, MongoDB, and Mongoose were installed to accommodate JavaScript programs and databases. The deployment on the AWS platform succeeds at the end of the first semester and users can get access to the website using a specific domain instead of running them in a local environment. However, the services for dynamic applications on AWS are expensive. Thus, the deployment of this project was migrated to Heroku in the second semester.

To deploy on the Heroku platform, a key is generated from the Heroku website and included in the application. Then the application will be registered in Heroku by specifying the path of it in the Heroku setting. Heroku will be registered as one of the origins in the “git” tool and all the code besides the “node_modules” directory can be pushed to it. After all the dependencies have been successfully installed, the URL of the application can be checked with the “Heroku open” command. The default domain name was generated randomly by Heroku. Custom domain names can be set up by configuring both the application’s DNS provider to point at the DNS target provided by Heroku and the Heroku’s domain list to add the custom domain. These two configurations need to be performed for both subdomains and root domains.

3. Future Work

3.1 Frontend and Backend

As has been previously mentioned, this component-based frontend has a tree-shaped structure. The components closer to the root (high-level components) contain fewer labels. Thus, it is easy for future developers to add new modules with even huge code to the current project. Moreover, the many low-level components, such as the card-shaped vital sign monitor, are repeatedly used. Hence, it is also convenient for future developers to re-use the existing code. For example, if an extra vital sign or hardware mode is required to be displayed in the dashboard, the card-shaped components can be directly re-used and the process can be completed by adding only several lines of code.

Scenario Editor

There is currently a functional scenario selector running at the backend. However, the current scenarios can only be edited at the backend. Although there was previously a functional scenario editor completed by our team in the first semester (see Fig.15), the code was abandoned after the frontend framework is transferred from JQuery to React. Thus, a user-friendly scenario editor is potential demand for users who is not proficient in programming.

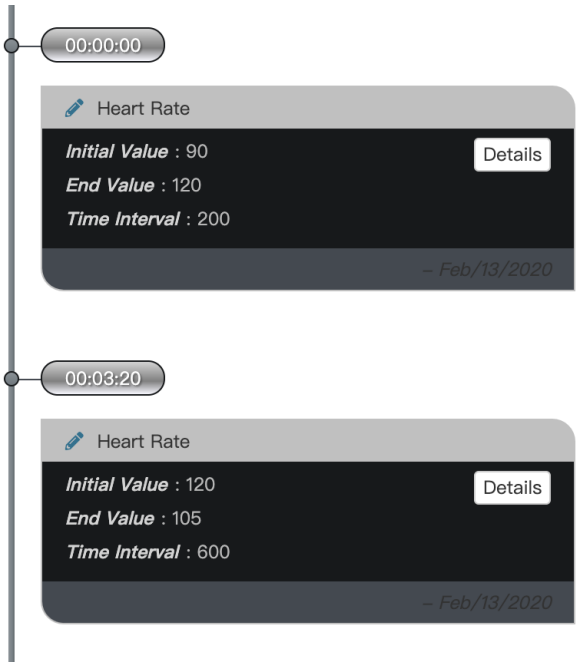


Fig.15 Scenario Editor Programmed Using JQuery

User Identity

Currently, only one identity (administrator) is defined in the user interface. For future work, the identity of students, who have only the authority to view vitals but get no access to setting operations, can be added to the current user management system. Fig.16 shows an existing component that can be used to extend this function.

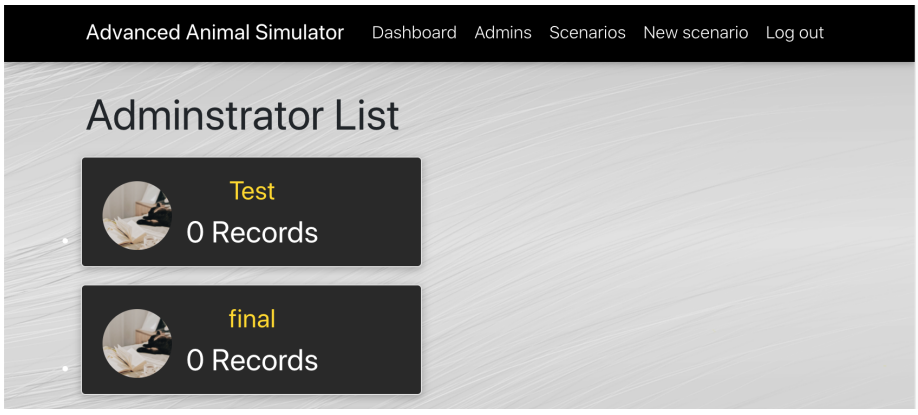


Fig.16 Administrator List

3.2 Integration through Database

This project of UI has been a functional entity until now. With respect to the integration of the entire robot dog (including the UI, hardware, mechanical components, and so on), the database is the intersection of the UI and the remaining parts. All the real-time data and modes are centrally managed in the database. Thus, the information exchange with the outside can be efficiently completed through the database operations.

4. Reference Documents for Future Developers

The reference documents for future developers are available in the Github repository (https://github.com/fl427/cornell-web-application). Important components and interfaces realized in the frontend and backend are specified for the purpose of future maintenance and extensions. The reference documents for the framework, package, and database are listed below.

React - frontend framework: https://reactjs.org/

Bootstrap - classic CSS framework: https://getbootstrap.com/

Material-UI: https://material-ui.com/

MDBootstrap - frontend components package: https://mdbootstrap.com/docs/react/

Express - backend framework: https://expressjs.com/

MongoDB Atlas - cloud platform of MongoDB: https://www.mongodb.com/cloud/atlas

Node.js - backend environment: https://nodejs.org/en/

AWS Elastic Beanstalk - backend deployment: https://aws.amazon.com/elasticbeanstalk/

AWS Simple Storage Service - static website deployment: https://aws.amazon.com/s3/

Heroku - free cloud computing platform: https://www.heroku.com/

References:

[1] Facebook inc. *"React: A JavaScript library for building user interfaces"*, https://reactjs.org/, 2020.

[2] MongoDB, inc., *"MongoDB: The database for modern applications"*, https://www.mongodb.com/, 2020.

[3] Amazon Web Services, inc., *"Amazon EC2"*, https://aws.amazon.com/ec2/, 2020.

[4] M. B. Jones, *"JSON Web Token"*, https://tools.ietf.org/html/rfc7519, 2015.