Dogtor Web Application: Veterinary Scheduling and GateKeeping Application

Object Design Document

ENGG4000 FR03X
Tash Zaman, Andre Aucoin and Jordan Mitchell

Table of Contents

1. Introduction

1.1 Object Design Trade offs

Much of the Dogtor system will be built from scratch. In the back-end, our team is utilizing pgAdmin4 to run our databases. We will make the database table and relationships ourselves, and will also design and implement the front-end ourselves. Though a large piece of the system will be developed by the Dogtor team, the team will make use of several existing UI components to provide an effective solution. The following is a list of UI components that our team will be outsourcing. Since we are adopting an agile method, we are currently still looking for the best components, but the following is the list plan for finding UI components for:

- Calendar
- Tables
- Password Encryptions

For the Dogtor system, we are aiming to build as many functional components as possible. Our team uses the parts designated above to efficiently make as many active features as possible, within the given timeframe.

1.2 Interface Documentation Guidelines

For clarification, our system is divided as follows: the front-end will consist of pages that use recurring components. Each page will be displayed at the endpoints setup in the index.js file. These endpoints are the same endpoints within the back-end where the API directs the program to our backend.js file which currently contains all of the functionality for all of the web pages in the Dogtor system.

- Comments will be written above each method to provide documentation on its purpose.
- Comments will be written at the top of each page to provide documentation on its purpose.
- Each method will receive documentation from the developer who implements it to include its purpose, the input values and what gets returned upon calling the method.
- Variable names should not start with a capital letter.
- Function names should not start with a capital letter.
- A page within the front end will start with a capital letter
- Function names must be clear about the purpose of what the function accomplishes.
- Variable names must be clear about the value that is being stored and what it means.
- Page names must be clear about the purpose of the page and what occurs in the UI.
- Errors will be dealt by the return values sent by the functions in the front-end by the back-end.

- If a variable name is more than a single word, CamelCasing must be used (example: reallyGoodExample).
- If a function name is more than a single word, CamelCasing must be used.
- If a component name is more than a single word, CamelCasing must be used.
- If a page name is more than a single word, CamelCasing must be used.
- For, while, and if statements that contain code that is more than a single line will utilize open and closing brackets.
- When entering a function,or a for, while, or if statement, the subsequent line must always be indented for any code inside the statement.

1.3 Glossary

Term	Definition
Pet	A pet that belongs to a specific client; the system will hold various details about the pet which will be used to book an appointment.
Client	The primary users of the system; They are users that book appointments for one to many pets.
Veterinarian (Vet)	A staff member with a veterinary degree (they may or may not be specialized). This staff member can perform any type of appointment.
Veterinary Technician (Technician)	A staff member that can only perform a particular set of appointments on pets on their own. These staff members can also assist vets during appointments, and have veterinary technician certification.
Veterinary Assistant (Assistant)	A staff member that performs both admin responsibilities and may assist veterinarians and veterinary technicians in their work.
Administrator (Admin)	Staff members responsible for monitoring the Dogtor web application. They are in charge of scheduling appointments, managing time-off for staff, rebooking appointments, and creating emergency appointments.

Appointment	An appointment is an interval of time that the client has booked that will allow a qualified staff member to perform veterinary medicine on their pet or will allow the client to ask the staff members questions concerning their pet's health. The pet being examined is specified in the appointment information.

1.4 References

[1] Allen H. Dutoit, Bernd Bruegge, "Object-Oriented Software Engineering, Second Edition" Prentice Hall, 2004.

2. Modules

The modules used by the Dogtor system will be utilizing several different software tools for both the front-end and the back-end. A more detailed list of the modules that our team will be using for the development of the Dogtor application is as follows:

- uuid-ossp is a module used to generate unique identifiers.
- pgcrypto is a module used to encrypt passwords as the user logs into the system.
- pg is a module used to set up and facilitate communication between the Dogtor application and our postgres server hosting our database.
- express is a module used to set up and facilitate communication between the application's front-end and back-end in the form of an API.
- body-parser is a module used to allow our team's API to parse incoming request bodies before the request is processed.
- http is a module used to create Dogtor's express server - to handle communication between the front-end and the back-end.
- axios is a promised-based http client used by our team to test endpoints in the App.test.js file.
- jest is our testing framework being used. All jest tests are contained in App.test.js.

Important Notice: since our team is currently in the midst of building the system, not all of the packages have been decided on. This list will continue to grow as development continues.

3. Class Interfaces Glossary

react

- The react package allows for a reliable way to create user-interfaces (all components are systematically organized.)
- React provides several APIs for manipulating individual elements and components.
- React provides hooks which allow for state use and other react features without utilizing classes.

react-dom

- react-dom provides modules which are specifically utilized in client and server applications.
- DOM is an acronym for Document Object Model which is just a structured representation of HTML elements in an application.
- react-dom allows web applications to dynamically update elements in real-time.

react-router-dom

- react-router-dom enables client side routing which is useful for moving between different pages.
- Client side routing allows applications to update the URL from a link click without making another request for another document from the server.
- React-router-dom allows for a more responsive experience as the browser does not need to request a whole new document to re-evaluate HTML/CSS and JavaScript assets for the next page.

styles.css

- A CSS style sheet that is applied to all of the webpages for a uniform UI design.
- CSS style sheets consist of different classes which can be called from a component to apply pre-set stylings.
- Style.css also removes the necessity to add inline styling for specific components.

Home.jsx

- The Home.jsx webpage is the first page that users are routed to when they go to the web application.
- The Home.jsx page is dependent on the header.jsx and footer.jsx UI components. It is also dependent on the react package and the react-router-dom package.
- The Home.jsx page has no public attributes.
- The Home.jsx page has the following public operations:

```
function Home() {
```

- Possible exceptions:
    - Can’t resolve object types if appropriate packages and UI components are not imported.

Login.jsx

- The Login.jsx webpage gets routed to from the Home.jsx webpage and allows the user to login to the system.
- The Login.jsx page is dependent on the header.jsx and footer.jsx UI components. It is also dependent on the react package.
- The Login.jsx page has no public attributes.
- The Login.jsx page has the following public operations:

```
function Login() {
const handleSubmit = (event) => {
const renderErrorMessage = (name) =>
const renderForm = (
```

- Possible exceptions:
    - Can’t resolve object types if appropriate packages and UI components are not imported.

Welcome.jsx

- The Welcome.jsx webpage gets routed to from the Login.jsx webpage and allows the user to navigate to differents parts of application (such as the scheduler and registration sections)
- The Welcome.jsx page is dependent on the header.jsx and footer.jsx UI components. It is also dependent on the react package.
- The Welcome.jsx page has no public attributes.
- The Welcome.jsx page has the following public operations:

```
function welcome() {
```

- Possible exceptions:

- Can’t resolve object types if appropriate packages and UI components are not imported.

Scheduler.jsx

- The Scheduler.jsx web page allows the user to schedule an appointment based on the availability of doctors/technicians.
- The Scheduler.jsx page is dependent on the header.jsx and footer.jsx UI components. It is also dependent on the react package and the external react-big-calendar and moment packages.
- The Scheduler.jsx page has no public attributes.
- The Scheduler.jsx page has the following public operations:

```
class App extends Component {

render() {
```

- Possible exceptions:
    - Can’t resolve object types if appropriate packages and UI components are not imported.

Header.jsx

- The Header.jsx class is a component that is appended to the top of every webpage that the user accesses
- The Header.jsx page is dependent on the react package
- The Header.jsx page has no public attributes.
- The Header.jsx page has the following public operations:

```
function Header() {
```

- Possible exceptions:
    - Can’t resolve object types if the appropriate package is not imported.

Footer.jsx

- The Footer.jsx class is a component that is appended to the bottom of every webpage that the user accesses
- The Footer.jsx page is dependent on the react package
- The Footer.jsx page has no public attributes.
- The Footer.jsx page has the following public operations:

```
function Footer() {
```

- Possible exceptions:
    - Can't resolve object types if the appropriate package is not imported.

backend.js

Contains all of the logic used to create, read, update, and delete data within the database (hosted on a postgres server). This file communicates with the API, contained in index.js. As this file continues to grow due to new features being added - it may be split up into more files to improve readability.

imports : pg

function login(u, p, res);

function schedule(details, date, vet, tech, pet, cust, compla, res);

index.js

This file contains Dogtor's express api. The file uses body-parser, http, and the backend.js file to facilitate communication from the front-end to the back-end, and vice-versa.

imports: express, body-parser, http, and backend.js

app.route('/check').post((req, res));

app.route('/appointment').post((req, res));

App.test.js

A file containing all jest tests used to test the back-end of the system.

imports: axios, index.js, and pg

test('Adding appointment', async ());

async function addAppointment();

dataBase

Contains all of the SQL statements required to set up the postgres database. This file also includes the insertion of some test values.

extensions: uuid-ossp

```
CREATE TABLE admins (

        admin_id uuid DEFAULT uuid_generate_v4 (),

        admin_name VARCHAR NOT NULL,

        admin_password VARCHAR NOT NULL,

        PRIMARY KEY(admin_id)

);


CREATE TABLE clients(

        client_id uuid DEFAULT uuid_generate_v4 (),

        client_username VARCHAR NOT NULL,

        client_password VARCHAR NOT NULL,

        client_notes VARCHAR,

        PRIMARY KEY(client_id)

);


CREATE TABLE pets(

        pet_id uuid DEFAULT uuid_generate_v4 (),

        pet_name VARCHAR NOT NULL,

        pet_appointements_notes TEXT [],

        owner_id uuid,

        pet_age int,

        pet_weight int,
        pet_sex VARCHAR,
        pet_neutered BOOLEAN,
        who_performed_last VARCHAR,
        last_date date,
        pets_behaviour TEXT,
        pet_active BOOLEAN,
        PRIMARY KEY(pet_id),
        FOREIGN KEY (owner_id) REFERENCES "clients" (client_id)
);

CREATE TABLE vets(
        vets_id uuid DEFAULT uuid_generate_v4 (),
        vets_username VARCHAR NOT NULL,
        vets_password VARCHAR NOT NULL,
        vets_skills text[],
        vets_speciality VARCHAR,
        vets_start_time time,
        vets_end_time time,
        PRIMARY KEY(vets_id)
);

CREATE TABLE technicians(
        technicians_id uuid DEFAULT uuid_generate_v4 (),
        technicians_username VARCHAR NOT NULL,
        technicians_password VARCHAR NOT NULL,
	technicians_skills text[],
	vets_speciality VARCHAR,
	technicians_start_time time,
	technicians_end_time time,
	PRIMARY KEY(technicians_id)
);

CREATE TABLE veterinary_assistant(
	veterinary_assistant_id uuid DEFAULT uuid_generate_v4 (),
	veterinary_assistant_username VARCHAR NOT NULL,
	veterinary_assistant_password VARCHAR NOT NULL,
	veterinary_assistant_skills text[],
	veterinary_assistant_start_time time,
	veterinary_assistant_end_time time,
	PRIMARY KEY(technicians_id)
);


CREATE TABLE appointment(
	appointment_id uuid DEFAULT uuid_generate_v4 (),
	appointment_date DATE,
	appointment_details VARCHAR(500),
	assigned_vets_id uuid,
	assigned_technicians_id uuid,
	assigned_pets_id uuid,
        assigned_Client_id uuid,

        chief_complaint VARCHAR,

        FOREIGN KEY (assigned_Client_id) REFERENCES "clients" (client_id),

        FOREIGN KEY (assigned_pets_id) REFERENCES "pets" (pet_id),

        FOREIGN KEY (assigned_technicians_id) REFERENCES "technicians"
(technicians_id),

        FOREIGN KEY (assigned_vets_id) REFERENCES "vets" (vets_id),

        PRIMARY KEY(appointment_id)

);


CREATE TABLE blocks(

        block_id uuid DEFAULT uuid_generate_v4 (),

        blockInt int,

        assigned_appointment_id uuid,

        FOREIGN KEY (assigned_appointment_id) REFERENCES "appointment"
(appointment_id),

        PRIMARY KEY(block_id)

);


INSERT INTO admins(admin_name,admin_password) VALUES
('AndreAucoin',crypt('password', gen_salt('md5')));

INSERT INTO clients(client_username, client_password) VALUES
('JohnDuck',crypt('password', gen_salt('md5')));

INSERT INTO vets(vets_username, vets_password,vets_skills ) VALUES ('Donald
Duck',crypt('password', gen_salt('md5')), null);

INSERT INTO technicians(technicians_username, technicians_password,technicians_skills )
VALUES ('Scrooge Mc Duck',crypt('password', gen_salt('md5')), null);
```