Dogtor Web Application: Veterinary Scheduling and GateKeeping Application

Software Design Document

ENGG4000 FR03X

Tash Zaman, Andre Aucoin and Jordan Mitchell

Table of Contents

1. Introduction

1.1 Purpose of the System

The Dogtor web application is a tool designed to allow veterinary clients to book appointments via the web or by calling or visiting their local veterinary clinic. The tool displays donation opportunities and resourceful links to the user, once the first log in, and the user can navigate to view their pet's information, adjust their settings, receive important notifications, and book and request a cancellation of an appointment. The appointment process for the user highlights the best veterinarian suited for the type of appointment, which is defined by the specialization of a vet. The application also aims to be more customizable than its competitors - allowing the office administrator to customize the scheduling system that suits their clinic best and follows the particular practices of the location of the vet clinic. All staff members (veterinarians, veterinary technicians, and office administrators) can view their schedules and request time off using the application. The Dogtor web application focuses on ease of use for its large demographic, and provides a more customizable and efficient solution to the organization of veterinary clinic appointments.

1.2 Design Goals

The Dogtor web application is a prototype that has the potential to be incorporated into a more advanced product at a later time. Due to the app being user facing to both our clients and their veterinary clients - our team will focus on following universal design principles to the best of our abilities. Our goal will be to design our app in an inclusive manner - taking into account as many demographics as possible that could potentially use our application. Our team also aims to make the final design of the Dogtor web application as customizable as possible. This was discussed with our client, and customization was found to be the biggest downfall of the current system that they are using.

As the vet clinic handles sensitive information about the veterinary clinic's clientele, another design goal for our system's design is to prioritize securing user information. This can be done through encryption and creating session tokens to ensure that users can only see the information that they are allowed to see. Finally, our team's last design goal is to create an application that is reliable and has high availability. Pet emergency appointments can happen at any hour of the day, so the application must have a high availability with a low number of crashes due to bugs within the code.

1.1 Definitions, Acronyms and Abbreviations

1. Dogtor: The working title of the project.

2. Technician Data Repository: A data store that holds all of the information regarding veterinary technicians and what type of specialties they have.

3. User Data Repository: A data store that holds all of the user's information and contains a list of appointment information that relates to them.

4. Veterinarian Data Repository: A data store that holds all of the information regarding veterinarians and identifies their current specialties in terms of what they have studied in school.

5. Pet Data Repository: A data store that holds the information pertaining to pets of veterinary clients, such as medical history, surgeries performed, weight and any other information that might be relevant to the veterinarian.

6. Appointment Data Repository: A data store that holds all of the appointments that have been booked.

7. Appointment-Type Data Repository: A data store that holds all of the information in terms of the type of appointment that the clinic can perform and who can perform each type of appointments (veterinarians, veterinary technicians or both).

8. Model View Controller (MVC): An architectural pattern used in software development commonly used for developing user interfaces. This divides the software into three elements: the model which contains the system's data, the view which is the user interface and the controller which connects the model and the view allowing the view to present data that is held within the model. This architectural pattern decouples the user interface from the system's business logic and data.

2. Current Software Architecture

Our team has reached out to a local veterinary clinic, Florenceville Veterinary Clinic, and have asked what software they are currently using to schedule and organize their appointments. Currently, the clinic is using a software tool called CornerStone, which can be used as an appointment scheduling tool. Currently, the top features of this software are that it has diagnostic integration, scheduling, inventory, EMR (an electronic medical record system), compliance assessment, imaging integration, invoicing, reporting, and provides a white to track patient status and upcoming tasks
[https://www.idexx.com/en/veterinary/software-services/cornerstone-software/cornerstone-featur

es/]. Our team discussed with Florenceville Veterinary Clinic's office administrator about potential features that we could include in our application that would improve what CornerStone has achieved thus far. She said that it would be ideal to include more customization within the proposed application and that allowing users to have more control over booking appointments would avoid some client-staff conflict when clients cannot book appointments when they desire. She also described how CornerStone can be a bit confusing at times. This is why our team is prioritizing user experience and system availability.

3. Proposed Software Architecture

3.1 Overview

Our proposed system will be utilizing the Model View Controller architecture. The following explanation will go in depth describing each element's functionality.

- **Model:** The backend of our system will be the model. Our model will be designed using NodeJS - which, in turn, will facilitate interactions between the backend code and the database. Each page will dictate how the user can interact with the model and will perform various tasks that will be placed within functions.

- **View:** In our system we will have three different views of the system: veterinarian & technician view, system administrator view and user view.
    - Veterinarian View: This view is for the Veterinarian whose view will be composed of their schedule and the various edits screens.
        - *Login*: Interface dedicated to log a user into the system.
        - *Main Menu*: Interface which will display a bulletin board holding resources such as donation opportunities and adoption services.
        - *View Schedule*: Interface which allows Veterinarians to view their appointment schedule and to edit it if needed.
        - *Request/Cancel Time Off*: Interface which allows Veterinarians to create/cancel a time off request.
        - *View Pets Profile*: Interface which allows Veterinarians to view the information about any pets pertaining to a client.
        - *Edit Pets Profile*: Interface which allows Veterinarians to edit information about any pets pertaining to a client.
        - *Edit Applications Settings*: Interface which allows Veterinarians to edit settings within the application.

- Technician View: This view is for the Technician whose view will be composed of their schedule and the various edits screens.
    - *Login*: Interface dedicated to log a user into the system.
    - *Main Menu*: Interface which will display a bulletin board holding resources such as donation opportunities and adoption services.
    - *View Schedule*: Interface which allows Veterinary Technicians to view their appointment schedule and to edit it if needed.
    - *Request/Cancel Time Off*: Interface which allows Veterinary Technicians to create/cancel a time off request.
    - *View Pets Profile*: Interface which allows Veterinary Technicians to view the information about any pets pertaining to a client.
    - *Edit Pets Profile*: Interface which allows Veterinary Technicians to edit information about any pets pertaining to a client.
    - *Edit Applications Settings*: Interface which allows Veterinary Technicians to edit settings within the application.

- Vet Client View: This view is for the vet client whose view will be composed to book appointments and edit pets information.
    - *Login*: Interface dedicated to log a user into the system.
    - *Main Menu*: Interface which will display a bulletin board holding resources such as donation opportunities and adoption services.
    - *Edit Pets Profile*: Interface which allows Veterinarians to edit information about any pets pertaining to a client.
    - *Cancel Appointments*: Interface which allows vet clients to request to cancel appointments.
    - *Create Regular Appointments*: Interface which allows vet clients to book appointments.
    - *Edit Applications Settings*: Interface which allows users to edit settings within the application.

- Office Administrator: This view is for the office workers who will be using the system daily.
    - *Login*: Interface dedicated to log a user into the system.
    - *Main Menu*: Interface which will display a bulletin board holding resources such as donation opportunities and adoption services.
    - *Approve Time Off Request*: Interface dedicated to viewing and approving time off requests from staff members.
    - *Edit Bulletin Board*: Interface dedicated to allow the Office Administrator to edit the information on the bulletin board (Main Menu).

        - *View Schedule*: Interface which allows Office Administrators to view their schedule, all staff schedules and to edit either of them if needed.
        - *Request/Cancel Time Off*: Interface which allows the Office Administrator to create/cancel a time off request.
        - *View Pets Profile*: Interface which allows Office Administrators to view the information about any pets pertaining to a client.
        - *Edit Applications Settings*: Interface which allows Office Administrators to edit settings within the application.
        - *Cancel Appointments*: Interface which allows Office Administrators to formally cancel appointments (via cancel request from a client or by verbal consent).
        - Create Appointments: Interface which allows Office Administrators to create any type of appointment (emergency and regular are both valid).

- **Controller:** In our project our controller will be our API, which is run using NodeJS. This will be implemented using Express, and will facilitate communication between the Model (implemented using NodeJS - interacts with Postgres) and the View (implemented using React). The API will process the input that it receives from the view and will invoke the appropriate functions in the model.

3.2 Subsystem Decomposition

Our system will be split into several subsystems each of which will have dedicated functionality. The tasks performed by these subsystems will include scheduling logic, information transformation logic and retrieve information logic.

- **Scheduling Logic:** The main purpose of our project is to design a system which simplifies the act of scheduling appointments. Due to the vast number of different appointment types our logic will function as follows.
    - *Vet Client*: When a Vet Client attempts to book an appointment they will have to answer specific questions which will allow the logic to determine which appointment type needs to be booked and who they can book the appointment with. The different appointment types will be determined by the administrators and inputted into the system. Once the logic determines which type of appointment to book the logic will provide a streamlined way to book the appointment.
    - *Office Administrator*: When an Office Administrator attempts to book an appointment, they don't need to fill out a survey like the vet client does since they already have an idea of what type of appointment needs to be booked by the information given to them by the client and/or a staff member. They will be able

to utilize the logic to book all appropriate appointments and will also be able to reschedule appointments that must be pushed due to an emergency.

- **Edit Information Logic:** Our system works with a wide variety of information such as vet information, pet information, and customer information so we need to provide various forms of logic to handle these categories of data. The following logic will be implemented for various actors to edit information.
    - *Vet Clients* should only have access to edit information that pertains to them and their pets.
    - *Vets* should have access to edit information that pertains to them and they should be able to edit any client's pet information regarding appointments and results.
    - *Office Administrators* should have access to edit all information across all actors.
    - *Technicians* should have access to edit information that pertains to them and they should be able to edit any client's pets regarding appointment and results.

- **Retrieve Information Logic:** Our team is dealing with a wide variety of information in this system. Due to this fact, we must have various forms of retrieval logic in order to retrieve all the information that we need. Here is the retrieval logic that will be implemented.
    - Retrieve all vets
    - Retrieve all technicians
    - Retrieve all users
    - Retrieve all pets
    - Retrieve all available type of appointment
    - Retrieve schedule
    - Retrieve vet
    - Retrieve technician
    - Retrieve user
    - Retrieve pet

3.3 Hardware/Software Mapping

Our team will be using NodeJS and React to design and implement the system. In terms of constraints, the system will initially be built to handle a single user at a time. The number of users within the system will increase as the system reaches the end of the implementation phase. Since we want our system to be widely used by a variety of clients, we will build a simple user interface which will be accessible and is easy to use. Our system will be web-based, which implies that the system can be used anywhere with internet access and is not restricted to a single veterinary clinic. The customization will be unique across each clinic, meeting each of their unique needs and standards.

3.4 Persistent Data Management

For this project, relational databases/repositories will be used to store user information. This is because the data has to be easily transferable between devices while also being secure which other data management systems may not be able to achieve as effectively.

- **Technician Data Repository**
  A data store that holds all of the information regarding veterinary technicians and what type of specialties they have.

- **User Data Repository**
  A data store that holds all of the user's information and contains a list of appointment information that relates to them.

- **Veterinarian Data Repository**
  A data store that holds all of the information regarding veterinarians and identifies their current specialties in terms of what they have studied in school.

- **Pet Data Repository**
  A data store that holds the information pertaining to pets of veterinary clients, such as medical history, surgeries performed, weight and any other information that might be relevant to the veterinarian.

- **Appointment Data Repository**
  A data store that holds all of the appointments that have been booked.

- **Appointment Type Data Repository**
  A data store that holds all of the information in terms of the type of appointment that the clinic can perform and who can perform each type of appointments (veterinarians, veterinary technicians or both).

3.5 Access Control and Security

User access to the system will require system authentication. In our use case, we will have three different types of users: Admin, Veterinarian/Veterinary Technicians, and Clients. If the user already has an account, then they will be prompted to login; otherwise, users will have to go to the hospital in-person to set up an account. Only Office Administrators can create new accounts for users.

	Login	**Profile Creation**	**Profile Editing**	**Knowledge Base**	**Repository List**	**Search**
Admin	**Yes**	**Yes**	**Yes** (all users) read, write	**Yes** read, write	**Yes** read, write	**Yes** read, write
Customer	**Yes**	**No**	**Yes** (only self) read, write	**Yes** read only	**Yes** read only	**Yes** read, write
Veterinarian	**Yes**	**No**	**Yes** (only self) read, write	**Yes** read only	**Yes** read only	**Yes** read, write
Technician	**Yes**	**No**	**Yes** (only self) read, write	**Yes** read only	**Yes** read only	**Yes** read, write

- **Login:**
  Users will need to enter their username and password to enter the system.

- **Registration:**
  Can be done only by an Office Administrator; once an Office Administrator creates an account, the specified user will have access to the system.

- **Profile creation:**

  Office Administrators will create a profile per the user's suggestions as part of the Registration process.

- **Profile editing:**
  Office Administrators will have the rights to edit all the profiles of the users.
  Users will have the rights to only edit their own profiles.

- **Knowledge Base:**
  Office Administrators will have the rights to edit and use the rules inside the knowledge base.
  Users will only have the right to use that knowledge base when search is done.

- **Repository List:**
  Office Administrators will be able to edit and use the list.
  Users will only be able to use the list when doing the search.

- **Search:**
  Office Administrators and users will both have access to the search/recommendation function of the system.

3.6 Global Software Control

The Dogtor web application will utilize an event-driven, centralized control design. Much of the interaction between the frontend and the system model will be done through the API which will be used. The API will request the system to perform different actions based on what the user is trying to achieve.

System synchronization will be maintained by combining the MVC architecture and the observer design pattern. Once an action has been performed by the system, it will notify the API to update the user interface with the translated information.

All web-based applications are **concurrent** by default. There may be multiple users signed onto the system using different devices; the web application will account for multiple instances being run. Typically, concurrency is handled by the API so much of the work will be done internally. Due to the limitations of the project scope, multi-threading support will not be implemented; however, an ideal system would include it to improve performance quality. The scheduling subsystem will require some sequential ordering, conditional logic will be used to regulate function calls to ensure that the precise order is maintained.

3.7 Boundary Conditions

Initialization:

Upon start-up, the API will instantiate the user interface and wait for the user to login. The user must enter a username and password which will cross-reference with the user data repository. If there is a match then the user may proceed to the home page; however, if there is no match then an error will be thrown (input validation). The API will also retrieve all of the user's saved states from their previous session. No user can register themselves from the login page, account registration must be done by a user with admin privileges after they have logged in.

Termination:

The user can choose to logout of the system via the user interface. This will ensure that all ongoing states in the system have been saved accordingly so they can pickup where they left off again. The user can also terminate the system asynchronously by closing the application. This is less secure as there is a chance some processes may stop abruptly and cause data to be lost. All states and processes shall be moderated by the API.

Failure:

Internal errors will be communicated between subsystems via the API such that if a subsystem were to fail, then others who know about it can respond or terminate accordingly. Error handling shall be implemented such that the API knows what to look for. There is also the possibility for external errors such as power outages and operating systems crashes. Due to the randomness of these errors occurring, there unfortunately is no way to account for them. Simply restarting the system after a failure will be equivalent to initialization.