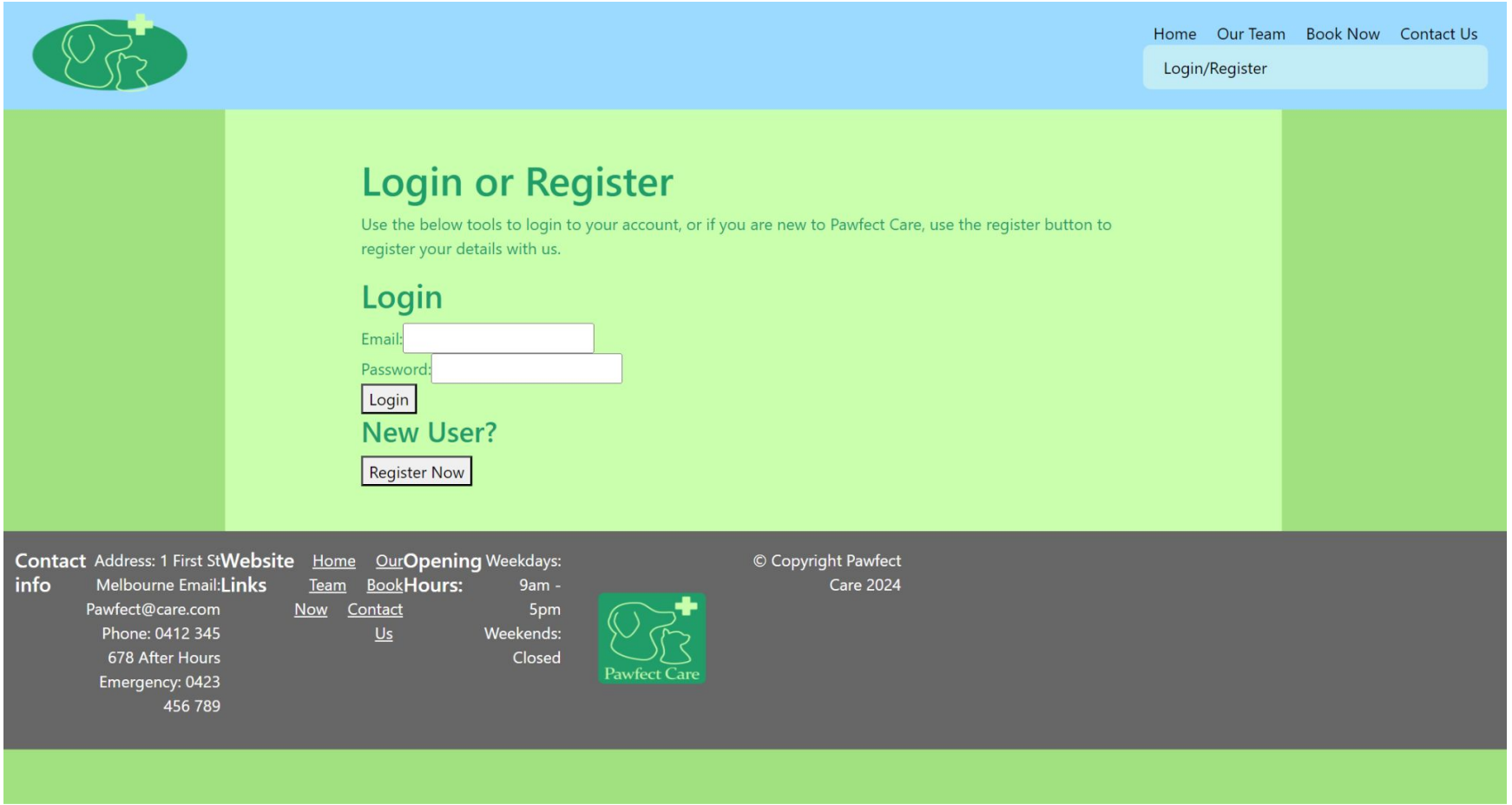
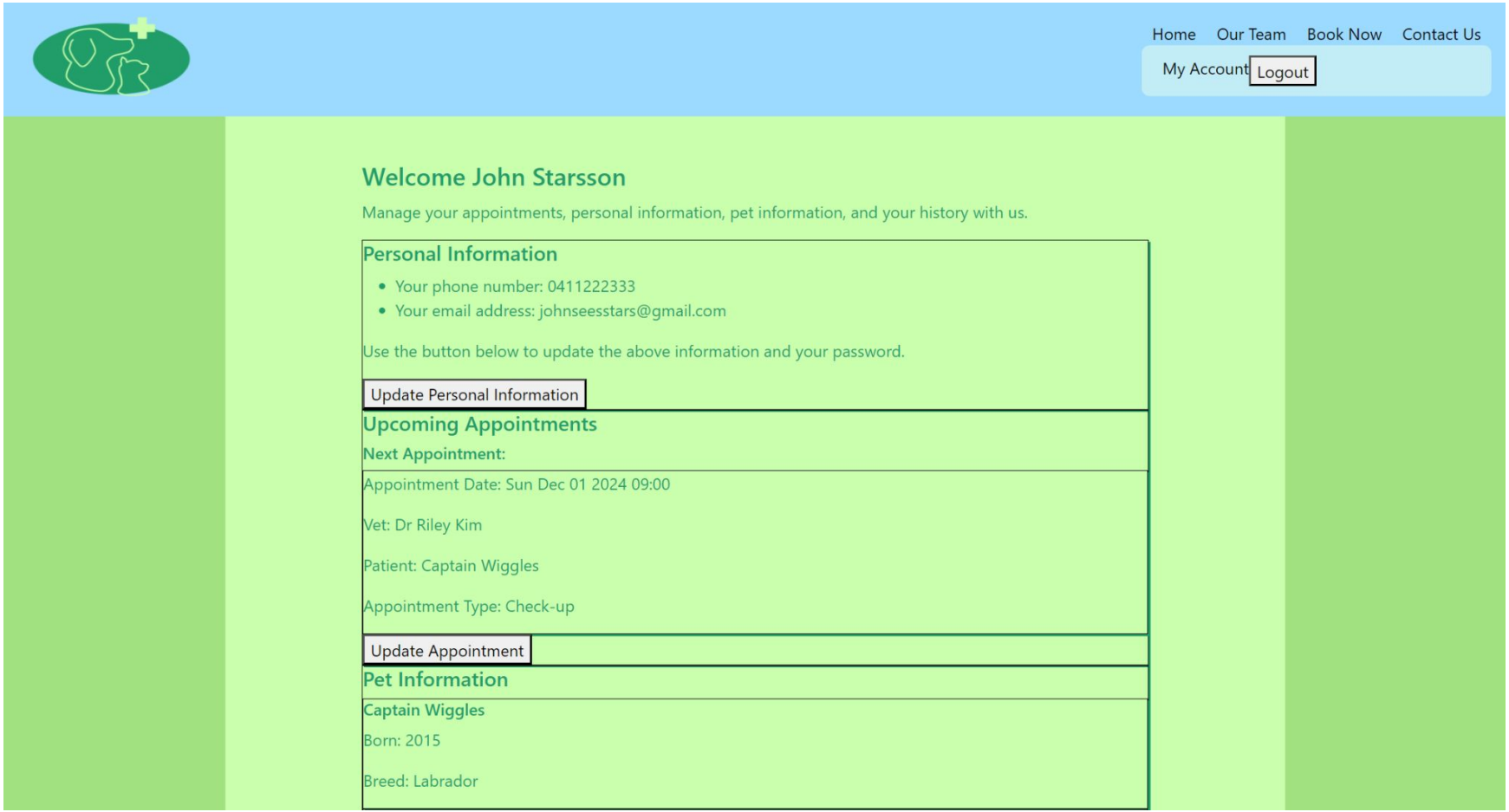
T3A2 Full Stack Application Presentation - *Pawfect Care*

Tom Tutone and Mike Sheppard

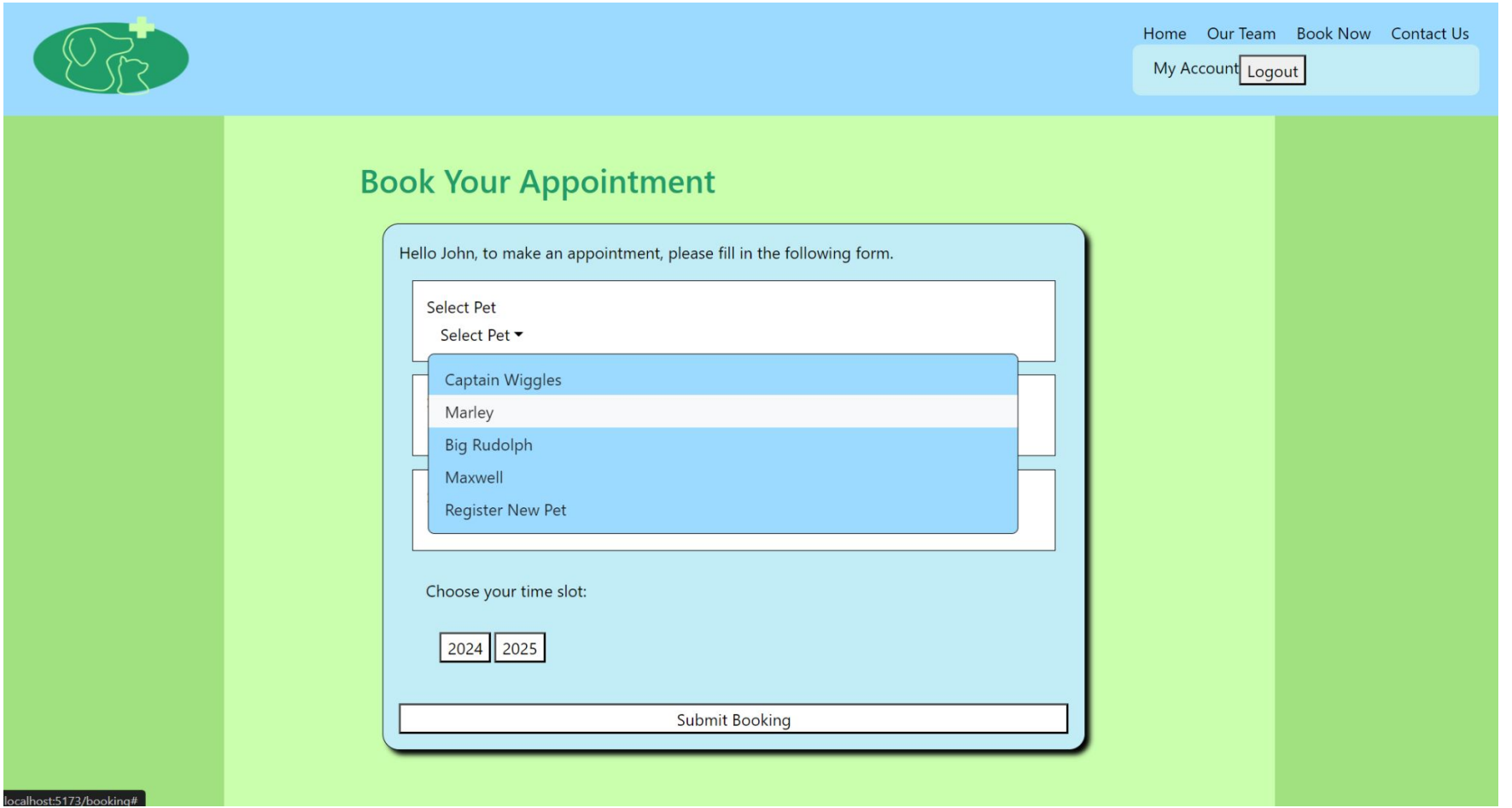
Website - Login Page



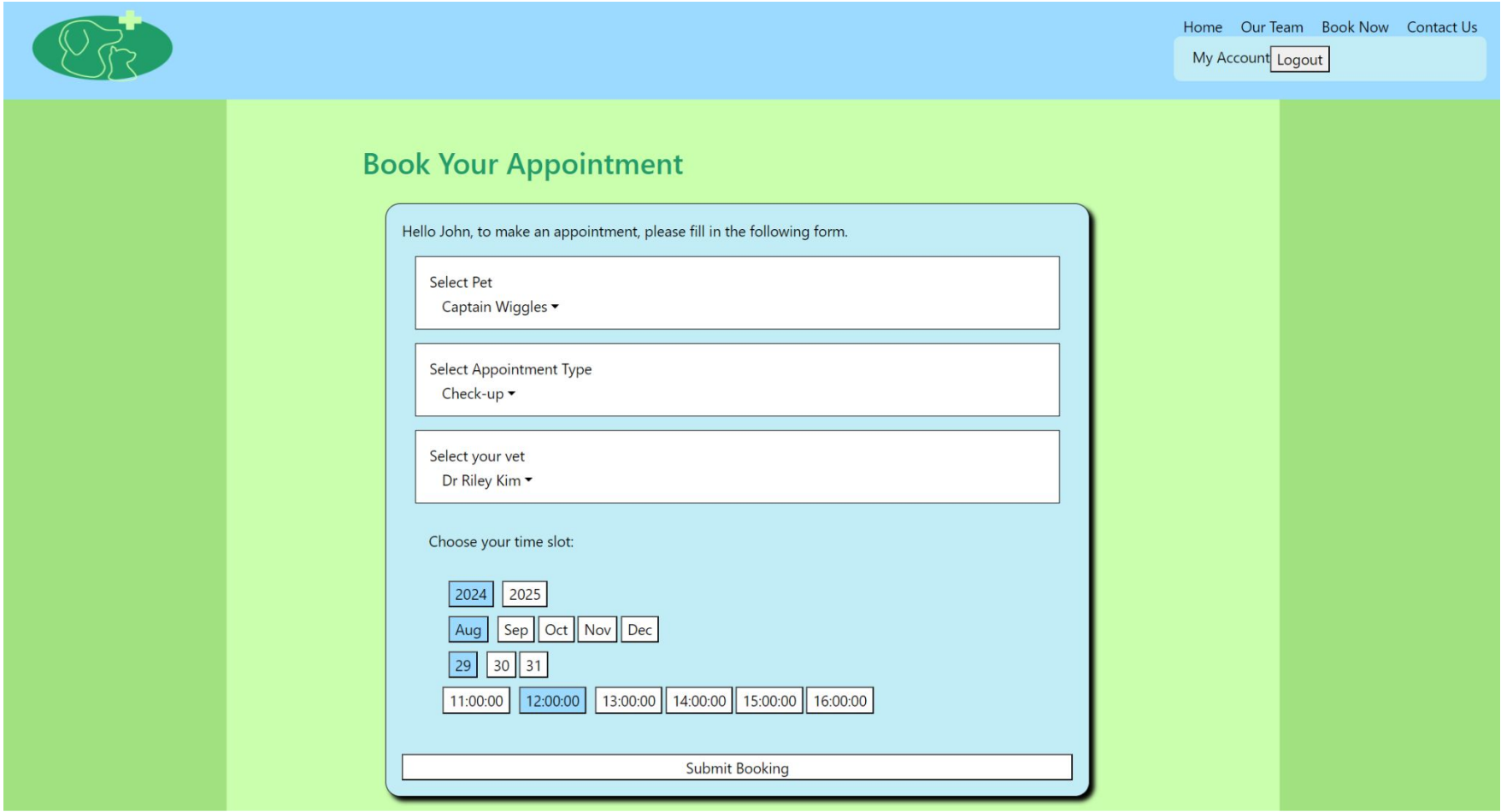
Website - My Account Page



Website - Select Pet Drop Down



Website - Select Time Buttons



Integral Code - Collection of States

States in Booking Tool

```
const token = sessionState((state) => state.token);
let userId
if (token) {
  userId = jwtDecode(token).userId;
}

const userData = sessionState((state) => state.userData);
const setUserData = sessionState((state) => state.setUserData);
const apiBase = sessionState((state) => state.apiBase);
const isAuthenticated = sessionState((state) => state.isAuthenticated);
const setIsAuthenticated = sessionState((state) => state.setIsAuthenticated);

const [petSelect, setPetSelect] = useState("");
const [serviceSelect, setServiceSelect] = useState("");
const [vetArray, setVetArray] = useState([]);
const [vetSelect, setVetSelect] = useState("");
const [timeSelect, setTimeSelect] = useState("");
const [errors, setErrors] = useState({});
const [submitSuccess, setSubmitSuccess] = useState(false);
```

Integral Code - Network Request

Vet Data Request

```
// Loads vets for populating the choices of vets and their appointments
// Load effect calls this upon the page loading
async function loadVets() {
  const response = await fetch(`${apiBase}/vets-list`, {
    headers: {
      "Content-Type": "application/json",
    },
  });

  if (!response.ok) {
    const errorData = await response.json();
    setVetArray('')
    console.log(errorData);
  }

  const retVets = await response.json();
  setVetArray(retVets);
}

// Calls loadVets on load
useEffect(() => {
  loadVets();
}, [userData]);
```

Integral Code - Network Data Display

Dropdowns

```
<div id="select-pet">
  <p>Select Pet</p>
  <SelectPetDropdown
    handlePetChange={handlePetChange}
    petSelect={petSelect}
  />
  {errors.petSelect && <p style={{ color: "red" }}>{errors.petSelect}</p>}
</div>
<div id="select-apptType">
  <p>Select Appointment Type</p>
  <SelectServiceDropdown
    handleServiceChange={handleServiceChange}
    serviceSelect={serviceSelect}
  />
  {errors.serviceSelect && (
    <p style={{ color: "red" }}>{errors.serviceSelect}</p>
  )}
</div>
<div id="select-vet">
  <p>Select your vet</p>
  <SelectVetDropdown
    handleVetChange={handleVetChange}
    vetSelect={vetSelect}
    vetArray={vetArray}
  />
</div>
```

Integral Code - Network Data Display

Pet Dropdown

SelectPetDropdown.jsx

src > components > Booking > SelectPetDropdown.jsx > SelectPetDropdown > userData.pets.map() callback

```jsx
import React from "react";
import { Dropdown } from "react-bootstrap";
import { useState } from "react";
import sessionState from "../../routes/store";
import RegisterPetForm from "../MyAcc/RegisterPetForm";

const SelectPetDropdown = ({ petSelect, handlePetChange }) => {
  const userData = sessionState((state) => state.userData);
  const [registerPet, setRegisterPet] = useState(false);

  return (
    <div>
      <Dropdown>
        <Dropdown.Toggle variant="secondary" id="dropdown-basic">
          {petSelect.petName || "Select Pet"}
        </Dropdown.Toggle>
        <Dropdown.Menu>
          {userData.pets ? (
            userData.pets.map((pet) => {
              return (
                <Dropdown.Item
                  onClick={() => {
                    setRegisterPet(false), handlePetChange(pet);
                  }}
                  key={pet.petName}>
                    {pet.petName}
                  </Dropdown.Item>
                );
              })
            ) : (
              <></>
            )}
            <Dropdown.Item onClick={() => {handlePetChange(`Register New Pet`), setRegisterPet
            (true);}}>
              Register New Pet
            </Dropdown.Item>
          </Dropdown.Menu>
        </Dropdown>
        {registerPet ? <RegisterPetForm /> : <></>}
      </div>
    );
};

export default SelectPetDropdown;
```

Integral Code - Generating Times / States

Calendar Dates States

BookingCalendar.jsx

src > components > Booking > BookingCalendar.jsx > BookingCalendar > genDays

```jsx
const BookingCalendar = ({
  vetArray,
  vetSelect,
  setTimeSelect,
  submitSuccess,
  setSubmitSuccess,
}) => {
  const monthList = [
    "Jan",
    "Feb",
    "Mar",
    "Apr",
    "May",
    "Jun",
    "Jul",
    "Aug",
    "Sep",
    "Oct",
    "Nov",
    "Dec",
  ];

  const timesList = [
    "09:00",
    "10:00",
    "11:00",
    "12:00",
    "13:00",
    "14:00",
    "15:00",
    "16:00",
  ];

  const today = new Date();
  const todayDay = today.getDate();
  const todayMonth = today.toString().slice(4, 7);
  const thisYear = today.getFullYear();
  const yearList = [thisYear, thisYear + 1];

  const [selectedYear, setSelectedYear] = useState("");
  const [displayMonths, setDisplayMonths] = useState("");
  const [selectedMonth, setSelectedMonth] = useState("");
  const [displayDays, setDisplayDays] = useState("");
  const [selectedDay, setSelectedDay] = useState("");
  const [displayTimes, setDisplayTimes] = useState("");
  const [timeButton, setTimeButton] = useState("");

```

Integral Code - Handling State Changes

Generating Month Buttons Based on State

```
// Handles clicks on year buttons
const yearClick = (year) => {
  if (year != selectedYear) {
    // Resets display values so all displayed buttons besides year are reset
    setDisplayMonths("");
    setDisplayDays("");
    setDisplayTimes("");
    setSelectedMonth("");
    setSelectedDay("");
    // Selected time for form, passed in
    setTimeSelect("");
    // Selected time button
    setTimeButton("");
    setSelectedYear(year);
  }
};

// Generates list of months for buttons to be generated from
const genMonths = () => {
  if (selectedYear) {
    if (selectedYear == thisYear) {
      for (let month in monthList) {
        if (monthList[month] == todayMonth) {
          setDisplayMonths(monthList.slice(month));
          break;
        }
      }
    } else {
      setDisplayMonths(monthList);
    }
  }
};

// Makes genMonths() dependent on selected year changing
useLayoutEffect(() => {
  genMonths();
}, [selectedYear]);
```

Integral Code - Generate Times (Loops, Conditional Statements)

Generating Times
Removing Invalid Times

```
const genTimes = () => {
  let timeBlocksArr = [];
  // Generate time objects for each time listed in available time list constant
  if (selectedDay) {
    for (let timeSlot of timesList) {
      timeBlocksArr.push(
        new Date(
          `${selectedDay} ${selectedMonth} ${selectedYear} ${timeSlot}`
        ).toString()
      );
    }

    // If selected day is today, remove times that are in the past
    if (selectedDay == todayDay) {
      for (let i in timeBlocksArr) {
        if (new Date(timeBlocksArr[i]) > today) {
          timeBlocksArr.splice(0, i - 1);
          break;
        }
      }
    }

    // Iterate through vets in vetArray
    for (let vet of vetArray) {
      // Check if the vet being considered is the select vet
      if (vet.vetName == vetSelect.vetName) {
        // Iterate through array of appointments in vet
        for (let appt of vet.appointments) {
          // Generate date string for individual appointment
          let apptDateString = new Date(appt.date).toString();
          // Check if appt date string is present in list of dates
          if (timeBlocksArr.includes(apptDateString)) {
            // Remove booked appointment time from generated times
            timeBlocksArr.splice(timeBlocksArr.indexOf(apptDateString), 1);
          }
        }
      }
    }
    setDisplayTimes(timeBlocksArr);
  }
};
```

Integral Code - Displaying Buttons (Conditional Statements)

Rendering Buttons if Truthy

```
return (
  <div>
    <div id="year-buttons">
      {yearList.map((year) => (
        <button
          onClick={() => {
            yearClick(year);
            setSubmitSuccess(false);
          }}
          key={year}
          className={
            selectedYear == year ? "selected-button" : "not-selected-button"
          }
        >
          {year}
        </button>
      ))}
    </div>
    <div id="month-buttons">
      {displayMonths ? (
        displayMonths.map((month) => (
          <button
            onClick={() => {
              monthClick(month);
              setSubmitSuccess(false);
            }}
            key={month}
            className={
              selectedMonth == month
                ? "selected-button"
                : "not-selected-button"
            }
          >
            {month}
          </button>
        ))
      ) : (
        <></>
      )}
    </div>
```

Integral Code - Network Request

Submitting Booking to Database

```
async function postNewBooking(e) {
  e.preventDefault();
  const isPetValid = validatePet();
  const isServiceValid = validateService();
  const isVetValid = validateVet();
  const isTimeValid = validateTime();

  if (isPetValid && isServiceValid && isVetValid && isTimeValid) {
    try {
      const response = await fetch(`${apiBase}/appointments`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`,
        },

        body: JSON.stringify({
          petId: petSelect._id,
          vetId: vetSelect._id,
          userId: userId,
          appointmentType: serviceSelect,
          date: timeSelect
        }),
      });

      if (!response.ok) {
        const errorData = await response.json();
        console.log(errorData);
        if (errorData["error/s"] == "invalid_token") {
          setIsAuthenticated(false);
        }
        throw errorData;
      }

      let submittedAppointment = await response.json();
```

```
      let submittedAppointment = await response.json();

      // Alter submitted Appointment object to made log in population
      submittedAppointment.petId = petSelect
      submittedAppointment.vetId = vetSelect
      setErrors((prevErrors) => ({ ...prevErrors, postError: "" }));
      setSubmitSuccess(true);
      setUserData({ appointments: submittedAppointment });
      console.log("post register user data, this console log is in MakeBookingForm.jsx",
      userData);
    } catch (err) {
      setErrors((prevErrors) => ({
        ...prevErrors,
        postError: err[err["error/s"]],
      }));
    }
```

Challenges - Global State

Login Function (Network Request)

```
const sessionState = create (
  (set) => ({
    login: async (email, password) => {
      try {
        const response = await fetch(`${apiBase}/users/login`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({ email, password }),
        })


        // Check if login promise.ok property is truthy - will be false is fetch fails
        if (!response.ok) {
          const errorData = await response.json()
          throw new Error(errorData.message || 'Failed to login: Please make sure your
          email and password are correct.')
        }

        // Convert login fetch promise to JSON obj
        const retToken = await response.json()
        console.log(retToken)

        // Set global state 'token', 'isAuthenticated'
        set({
          token: retToken.JWT,
          isAuthenticated: true,
          error: null,
        })

        // uId for passing to the call below to users/:id
        const uId = jwtDecode(retToken.JWT).userId

        // Call to user's specific end point, get their data
        const userIdGet = await fetch(`${apiBase}/users/${uId}`, {
          headers: {
            Authorization: `Bearer ${retToken.JWT}`,
            'Content-Type': 'application/json',
          }
        })

        // Check if promise.ok property is truthy - will be false is fetch fails
        if (!userIdGet.ok) {
          const errorUser = await userIdGet.json()
          throw new Error(errorUser.message || 'Failed to load user data')
        }
```

```
        // Convert retrieved login promise to JSON obj
        const retUserData = await userIdGet.json()

        // Set global state values
        set({
          error: null,
          userData: retUserData
        })

      }

      catch (error) {
          console.error("Login error:", error.message)
          set({
            token: null,
            isAuthenticated: false,
            error: error.message,
          })
      }
    },

    logout: () => {
      set({
        token: null,
        isAuthenticated: false,
        error: null,
        userData: {},
        publicApptData: null
      })
    },
  }),
  {
    name: 'loggedInData',
    storage: createJSONStorage(() => sessionStorage)
  },
),
)
```

Challenges - userData Updates

Updating Session State (Conditional)

```
const sessionState = create (
  persist (
  (set) => ({
      users: [],
      publicApptData: {},
      userData: {},
      setUserData: (newData) => {
        set((state) => ({
          userData: {
            ...state.userData,
            email: newData.email ? newData.email : state.userData.email,
            firstName: newData.firstName ? newData.firstName : state.userData.firstName,
            lastName: newData.lastName ? newData.lastName : state.userData.lastName,
            phNumber: newData.phNumber ? newData.phNumber : state.userData.phNumber,
            pets: newData.pets ? (newData.pets.length == 0 ? [] : [...state.userData.pets,
            newData.pets]) : state.userData.pets,
            appointments: newData.appointments ? (newData.appointments.length == 0 ? [] : [...
            state.userData.appointments, newData.appointments]) : state.userData.appointments
          }
        }))
      },
      token: null,
      isAuthenticated: false,
      setIsAuthenticated: (changeValue) => {
        set((state) => ({isAuthenticated: changeValue}))
      },
```

Challenges - Back End Error Handling (Database Operation)

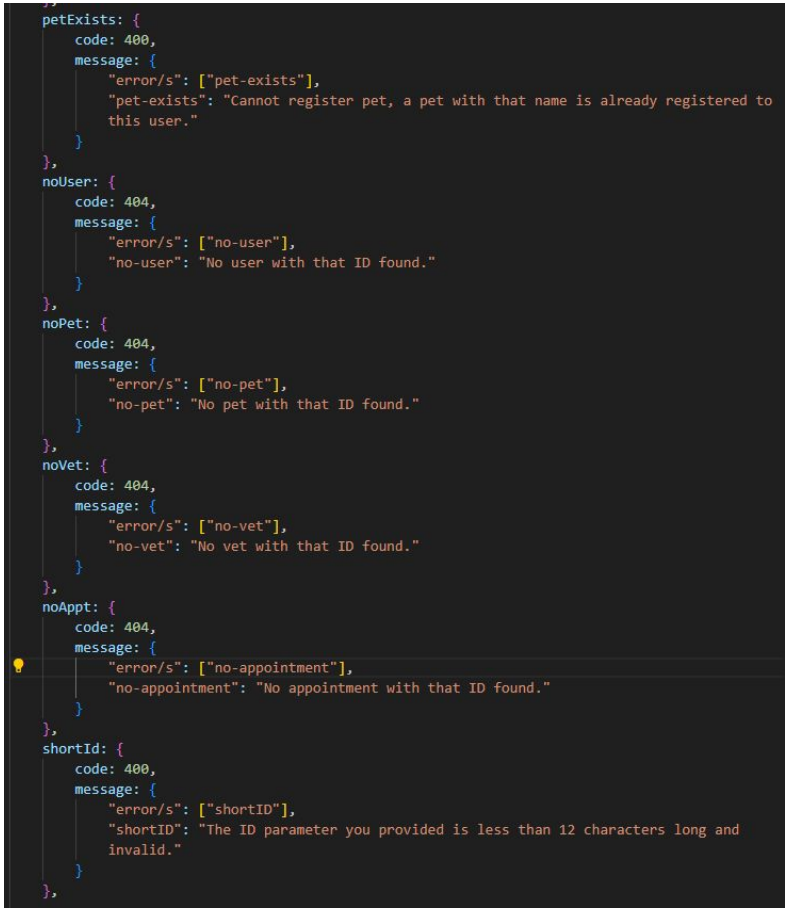
Creating a Pet / Retrieving a Pet

```
// Create a pet
router.post(`${petsPrefix}`, async (req, res, next) => {
    try {
        let { userId , isAdmin } = req.auth
        // Check if user is admin, otherwise sets UserId to JWT value
        if (!isAdmin) {
            req.body.userId = userId
        }

        // Check if pet with same name exists in DB registered to that user
        let petCheck = await Pet.findOne({userId: req.body.userId, petName: req.body.petName})
        if (petCheck) {
            throw customErrors.petExists
        }

        // Create a new pet object and add it to the DB
        const newPet = await Pet.create(req.body)

        // Retrieve User who registered pet
        let retUser = await User.findOne({_id: req.body.userId})

        // Add new pet to retrieved user
        retUser.pets.push(newPet._id)

        // Update user with new values
        let saveUser = await User.findOneAndUpdate({_id: req.body.userId}, {pets: retUser.
        pets}, {returnDocument: 'after'})

        // Respond to the client with the registered Pet instance
        res.status(201).send(newPet)
    }
    catch (err) {
        next(err)
    }
})
```

```
// Get single pet
router.get(`${petsPrefix}/:id`, async (req, res, next) => {
    try {
        if (req.params.id.length < 24) {
            throw customErrors.shortId
        }

        let { userId , isAdmin } = req.auth

        // Retrieve pet
        const pet = await Pet.findById(
            req.params.id
        ).populate({
            path: 'appointments',
            select: '-__v -petId',
            populate: [
                {
                    path: 'userId',
                    select: 'firstName lastName'
                },
                {
                    path: 'petId',
                    select: '-appointments -__v'
                }
            ]
        })

        if (pet) {
            if (isAdmin || pet.userId._id == userId ){
                res.send(pet)
            }
            else {
                throw customErrors.authError
            }
        } else {
            throw customErrors.noPet
        }
    }
    catch (err) {
        next(err)
    }
})
```

Challenges - Back End Error Handling

Custom Error Objects



```
petExists: {
    code: 400,
    message: {
        "error/s": ["pet-exists"],
        "pet-exists": "Cannot register pet, a pet with that name is already registered to
        this user."
    }
},
noUser: {
    code: 404,
    message: {
        "error/s": ["no-user"],
        "no-user": "No user with that ID found."
    }
},
noPet: {
    code: 404,
    message: {
        "error/s": ["no-pet"],
        "no-pet": "No pet with that ID found."
    }
},
noVet: {
    code: 404,
    message: {
        "error/s": ["no-vet"],
        "no-vet": "No vet with that ID found."
    }
},
noAppt: {
    code: 404,
    message: {
        "error/s": ["no-appointment"],
        "no-appointment": "No appointment with that ID found."
    }
},
shortId: {
    code: 400,
    message: {
        "error/s": ["shortID"],
        "shortID": "The ID parameter you provided is less than 12 characters long and
        invalid."
    }
},
```

Challenges - Back End Error Handling

Model Validation

```
const petSchema = new Schema({
    userId: {
        type: mongoose.Types.ObjectId,
        required: true,
        ref: 'User',
        validate: {
            validator: async function (id) {
                let user = await User.findById(id)
                if (user) {
                    return true
                }
                else {
                    return false
                }
            },
            message: props => `${props.value} is not a registered userID`
        }
    },
    petName: {type: String, required: true},
    birthYear: {
        type: Number,
        required: true,
        validate: {
            validator: async function (year) {
                let thisYear = (new Date).getFullYear()
                if (year >= 2000 && year <= thisYear) {
                    return true
                }
                else {
                    return false
                }
            },
            message: props => `${props.value} is an invalid birth year. Please enter a birth
            year after 2000 and not past the current year.`
        }
    },
    breed: {type: String, required: true},
    animalType: {type: String, required: true, enum: {values: ['dog', 'cat', 'other'],
    message: "animalType must be one of 'dog', 'cat' or 'other'."}},
    appointments: [{type: mongoose.Types.ObjectId, ref: 'Appointment'}],
});

const Pet = mongoose.model('Pet', petSchema)

export { petSchema, Pet }
```

Challenges - Back End Error Handling

Error Handling Middleware

Thanks for listening

Note to assessors:
Please see *T3A2-B Presentation Notes.pdf* as well.