



T3A2 Full Stack Application Presentation - *Pawfect Care*

Tom Tutone and Mike Sheppard

Website - Login Page



[Home](#) [Our Team](#) [Book Now](#) [Contact Us](#)

[Login/Register](#)

Login or Register

Use the below tools to login to your account, or if you are new to Pawfect Care, use the register button to register your details with us.

Login

Email:

Password:

New User?

Contact info

Address: 1 First St
Melbourne
Pawfect@care.com
Phone: 0412 345
678 After Hours
Emergency: 0423
456 789

Website Links

[Home](#)
[Team](#)
[Now](#)
[Contact Us](#)

Our Opening Hours:

Weekdays:
9am -
5pm
Weekends:
Closed



© Copyright Pawfect
Care 2024

Website - My Account Page



[Home](#) [Our Team](#) [Book Now](#) [Contact Us](#)

[My Account](#) [Logout](#)

Welcome John Starsson

Manage your appointments, personal information, pet information, and your history with us.

Personal Information

- Your phone number: 0411222333
- Your email address: johnseesstars@gmail.com

Use the button below to update the above information and your password.

[Update Personal Information](#)

Upcoming Appointments

Next Appointment:

Appointment Date: Sun Dec 01 2024 09:00

Vet: Dr Riley Kim

Patient: Captain Wiggles

Appointment Type: Check-up

[Update Appointment](#)

Pet Information

Captain Wiggles

Born: 2015

Breed: Labrador

Website - Select Pet Drop Down



[Home](#) [Our Team](#) [Book Now](#) [Contact Us](#)

[My Account](#) [Logout](#)

Book Your Appointment

Hello John, to make an appointment, please fill in the following form.

Select Pet

Select Pet ▼

Captain Wiggles

Marley

Big Rudolph

Maxwell

Register New Pet

Choose your time slot:

2024

2025

Submit Booking

Website - Select Time Buttons



[Home](#) [Our Team](#) [Book Now](#) [Contact Us](#)

[My Account](#) [Logout](#)

Book Your Appointment

Hello John, to make an appointment, please fill in the following form.

Select Pet
Captain Wiggles ▼

Select Appointment Type
Check-up ▼

Select your vet
Dr Riley Kim ▼

Choose your time slot:

2024 2025

Aug Sep Oct Nov Dec


29 30 31

11:00:00 12:00:00 13:00:00 14:00:00 15:00:00 16:00:00

Submit Booking

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 X
src > components > Booking > SelectPetDropdown.jsx > SelectPetDropdown > userData.pets.map() callback
1 import React from "react";
2 import { Dropdown } from "react-bootstrap";
3 import { useState } from "react";
4 import sessionState from "../../routes/store";
5 import RegisterPetForm from "../MyAcc/RegisterPetForm";
6
7 const SelectPetDropdown = ({ petSelect, handlePetChange }) => {
8   const userData = sessionState((state) => state.userData);
9   const [registerPet, setRegisterPet] = useState(false);
10
11   return (
12     <div>
13       <Dropdown>
14         <Dropdown.Toggle variant="secondary" id="dropdown-basic">
15           {petSelect.petName || "Select Pet"}
16         </Dropdown.Toggle>
17         <Dropdown.Menu>
18           {userData.pets ? (
19             userData.pets.map((pet) => {
20               return (
21                 <Dropdown.Item
22                   onClick={() => {
23                     setRegisterPet(false), handlePetChange(pet);
24                   }}
25                   key={pet.petName}>
26                   {pet.petName}
27                 </Dropdown.Item>
28               );
29             })
30           ) : (
31             <></>
32           )}
33           <Dropdown.Item onClick={() => {handlePetChange("Register New Pet"), setRegisterPet(true);}}>
34             Register New Pet
35           </Dropdown.Item>
36         </Dropdown.Menu>
37       </Dropdown>
38       {registerPet ? <RegisterPetForm /> : <></>}
39     </div>
40   );
41 };
42
43 export default SelectPetDropdown;
44
```

Integral Code - Generating Times / States

Calendar Dates States



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

4  const BookingCalendar = ({
5    vetArray,
6    vetSelect,
7    setTimeSelect,
8    submitSuccess,
9    setSubmitSuccess,
10 }) => {
11   const monthList = [
12     "Jan",
13     "Feb",
14     "Mar",
15     "Apr",
16     "May",
17     "Jun",
18     "Jul",
19     "Aug",
20     "Sep",
21     "Oct",
22     "Nov",
23     "Dec",
24   ];
25
26   const timesList = [
27     "09:00",
28     "10:00",
29     "11:00",
30     "12:00",
31     "13:00",
32     "14:00",
33     "15:00",
34     "16:00",
35   ];
36
37   const today = new Date();
38   const todayDay = today.getDate();
39   const todayMonth = today.toString().slice(4, 7);
40   const thisYear = today.getFullYear();
41   const yearList = [thisYear, thisYear + 1];
42
43   const [selectedYear, setSelectedYear] = useState("");
44   const [displayMonths, setDisplayMonths] = useState("");
45   const [selectedMonth, setSelectedMonth] = useState("");
46   const [displayDays, setDisplayDays] = useState("");
47   const [selectedDay, setSelectedDay] = useState("");
48   const [displayTimes, setDisplayTimes] = useState("");
49   const [timeButton, setTimeButton] = useState("");
50 }
```

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 if 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 if 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

```
seedjs  errorHandler.js  appointmentsRoutes.js  app.js
routers > errorHandler.js > errorHandler

4
5 function errorHandler (err, req, res, next) {
6   let retErrObj
7
8   // Handles mongoose errors with multiple errors at once
9   if ("errors" in err) {
10     let errKeys = Object.keys(err.errors)
11     retErrObj = {"error/s": errKeys}
12     for (let err1 of errKeys) {
13       if (err.errors[err1].reason) {
14         retErrObj[err.errors[err1].path] = `${err.errors[err1].reason}`
15       }
16       else {
17         retErrObj[err.errors[err1].path] = `${err.errors[err1].message}`
18       }
19     }
20     res.status(400).send(retErrObj)
21   }
22
23   // Handles mongoose errors with only one error
24   else if ("path" in err) {
25     retErrObj = {"error/s": [err.name]}
26     retErrObj[err.name] = String(err.reason)
27     res.status(400).send(retErrObj)
28   }
29
30   // Handles JWT auth errors
31   else if (err.code == 'credentials_bad_format' || err.code == 'credentials_required' ||
32     err.code == 'invalid token') {
33     retErrObj = {"error/s": [err.code]}
34     retErrObj[err.code] = err.inner.message
35     res.status(400).send(retErrObj)
36   }
37
38   // Handles custom errors I have coded, errors thrown are described in '../errorObjs.js'
39   else {
40     if (typeof(err.code) == 'number') {
41       retErrObj = {...err}
42       delete retErrObj.code
43       res.status(err.code).send(retErrObj.message)
44     }
45     else if (err.status) {
46       res.status(err.status).send(err)
47     }
48     else {
49       res.send(err).status(400)
50     }
51   }
52 }
```



Thanks for listening



Note to assessors:

Please see

T3A2-B Presentation Notes.pdf
as well.