Presentation Notes:

R14. Include an explanation of AT LEAST one of the key, essential, or otherwise integral pieces of code (such as a function or file) written for the core MVP features of the project.

Code explained should represent AT LEAST one of the following:

- a network request
- a database operation
- a loop
- a function call
- a conditional statement

MVP: Booking Feature

Comprised of:

1. Collection of states:
    a. Token, userData, setUserData, apiBase, setIsAuthenticated are all from global state (Zustand)
    b. The others are local states and setters for selections and a tracker of submission success
2. On page load, vets are loaded from endpoint '/vets-list'
    a. No JWT required, returns a list of vets and their appointments (only times not personal info)
    b. This is saved to local state for reference in the vets dropdown and appointment time generation
3. Drop down menus
    a. The drop downs are handed set selection functions
    b. Below each is a space for errors to be listed in red font which show up when the data is submitted
    c. Vet array is handed to vets to generate a live list of vets
    d. Vet array is handed to Booking calendar for the same reason
4. Pet drop down
    a. Shows a list of pets which is taken from the global state of user data
    b. Also contains a register pet section which when selected causes a popup allowing user to register a pet and prompts a login when the user clicks submit if they're not logged in
5. BookingCalendar States
    a. Contains a lot of states to track the selected values in the year, month, day and time buttons
    b. An array with lists for mapping to button contains this year and one ahead

6. Generate months
    a. On click of a year, all of the below buttons are reset
    b. Generate months triggers on any change in selected year
    c. Generate months checks if the current year is selected and removes all months before the current month
    d. The same takes place for days
7. Generate Times
    a. Most complex part of the function
    b. Refers to static list of times, generates a number of date objects with the selected year, month and date combined with the static times, saves them as a date string
    c. If the selected day is today, iterates through and removes all dates before the current time
    d. Iterates through lists of vet, finds the vet object with a vetName that matches the selected vet
    e. Cycles through that vet's appointments, converts each to a date object then to a string and compares them to the array of date strings genned for a day, on a match, removes that time from the list of times
    f. List of times is set to local state and the list is mapped for an array of times
8. Return statements
    a. When the list of years, months, days and times is generated, it is displayed so it doesn't crash when it changes/disappears
9. Post appointment part 1
    a. Validators check that values are selected for all categories
    b. A POST request is generated with the appropriate values set
    c. An error checker looks to see if the response is okay otherwise it adds the error to a list of errors which show below each dropdown
    d. If the error is an invalid JWT, such as might happen if the user is logged out but has their data in, which can happen because we're using persistent global state, global state of isAuthenticated is set to false which triggers a login popup for the user to log in within the MakeBookingForm return
10. Post appointment part 2
    a. If no errors are caught, the response is converted to a JSON
    b. The submitted pet and vet values are then converted to their respective objects rather than their objectId so that they can be displayed else where
    c. The altered values are then passed to the setUserData function which is a global state setter
    d. setSubmitSuccess is set to true which makes a success message render

R15. Include an explanation of AT LEAST one challenge or problem that occurred during the project's development, as well as how that challenge or problem was overcome.

Challenges or problems explained should be relevant to AT LEAST one of the following:

- a network request
- a database operation
- a loop
- a function call
- a conditional statement

Challenges - Global state implementation, back end error handling

11. Global State - login
    a. One trouble - userData is required all around the website, it was messy handing states back and forth neatly so we decided to implement Zustand
    b. We first set up a login function in global state
    c. This includes a fetch request with the users login data
    d. The returned object is a JWT which is used to set a token value for access around the site
    e. 11B - using the retrieved token, it is decoded for the usersId and this with the JWT are sent in another fetch request to access the user's data
    f. The usersData global var is set to the retrieved info
2. This led to the challenge of the setUserData function which could receive data neatly when a whole collection was retrieved but was more complex based on the different retrievals of new data such as registering a pet or registering a new user who has no data
    a. For this reason we had to write a complex collection of ternary operands
    b. Pets and appointments are similar, they check to see if new data passed to them exists, if it does, the data is checked to see if it is an array with a length of zero, such as if a user is newly registered and, if so, sets the value to a blank array. Otherwise, the old state is spread using the spread operand and the new data is added too. If no new pet or appointment data is passed to the function, such as if the user updates their email, it is set to the original value
3. Back end troubles error handling -
    a. One challenge on the backend was handling the variety of errors that could be thrown, each from different extensions and each formatted differently
    b. Our back end uses JWT tokens and mongoose, both which throw unique errors as objects which were hard to convert into a string which was descriptive and made sense to the user
    c. JWT errors are thrown on all but 3 routes so we had to handle those
    d. In addition, mongoose would throw a particularly obtuse error if the submitted iD parameter was too short so we had to make a custom error

for that, in addition we had to make errors to handle incorrect authorisation which would send the correct response code

4. Custom errors -
    a. These are custom errors which contain a description and error message
    b. These are for the error situations which are better to describe in plain terms such as no instance of a pet being found or a pet with a duplicate name attempted to be registered to the same user
5. Model Validation
    a. In addition, we needed control over the model errors
    b. Part of creating descriptive errors was creating custom validators because Mongo is noSQL but we wanted linked instances
    c. This meant writing messages which describe the issue such as a userId submitted to create a pet not existing
    d. These are then passed to an error handling middleware
6. Error Handler Middleware
    a. The errors are passed to a custom errorHandler that identifies which software generated the error based on what attributes are pressed in the error object
    b. For example, the mongoose errors are sent as an object containing all errors which are described within an attribute “errors”. The error handler iterates through this object, accesses the “path” attribute which we found describes the name of the incorrect value so we create an array with the error name, push it to it, and create objects with key-pairs of the path and message
    c. Otherwise we handle the different error types uniquely for our custom errors and set the error code as described in the error and send it to the user as a response
    d. Broadly this was trial and error, converting errors to strings and trying to identify attribute values
    e. In addition, referring to the documentation to learn how to create custom errors that are descriptive