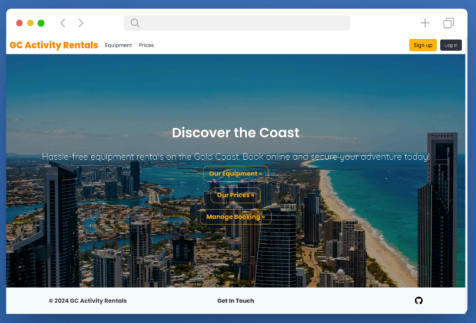
T3A2: Full Stack App

Xanthia Mason and Mohammed Hani



Our Full Stack Web Application

We were tasked by GC Activity Rentals to create a dynamic website that allowed users to view the equipment they offer, their prices, and allow them to seamlessley book online. In addition to this, it would allow the admin of the business to manage bookings and the equipment available for hire.

Key Functionality

1. User Sign Up and Login
2. Equipment and Price Listing
3. Booking System
4. Admin Functionality

Integral Piece of Code - Create a Booking

The main goal of the application was to streamline the process of booking equipment. As a result, one of the most integral pieces of code was the ability to create and store a booking in the database.

The Code Behind Create a Booking


```
// Create a new Booking
router.post('/bookings', verifyUser, async (req, res) => {
    try {
        const { equipment, quantity, startTime, hireOption } = req.body;
```

Example Request Body

```
{
  "equipment": "66ca701b1356d86e82a97ca3",
  "startTime": "2024-08-29T10:05:02.786Z",
  "hireOption": "66ca701b1356d86e82a97c93",
  "quantity": 2
}
```

Destructuring

```
        // Validate input fields
        if (!equipment || !quantity || !startTime || !hireOption) {
            return res.status(400).send({ error: 'Equipment ID, quantity, start time, and hire option ID are required.' });
        }
```

Conditional Statement

```
        // Parse startTime and check for validity
        const start = new Date(startTime);
        if (isNaN(start.getTime())) {
            return res.status(400).send({ error: 'Invalid start time format.' });
        }
```

Data Conversion

Conditional Statement

```
        // Fetch HireOption to get the hire length
        const hireOptionDoc = await HireOption.findById(hireOption);
        if (!hireOptionDoc) {
            return res.status(404).send({ error: 'Hire Option not found.' });
        }
```

Database Operation

Conditional Statement

```
        // Calculate endTime
        const end = new Date(start);
        end.setMinutes(end.getMinutes() + hireOptionDoc.length);

        if (start >= end) {
            return res.status(400).send({ error: 'End time must be after start time.' });
        }
```

Data Calculation

Conditional Statement

```
        // Retrieve equipment
        const equipmentSelected = await Equipment.findById(equipment);
        if (!equipmentSelected) {
            return res.status(404).send({ error: 'Equipment not found.' });
        }
```

Database Operation

Conditional Statement

```
88          return res.status(404).send({ error: 'Equipment not found.' });
89      }
90
91      // Check current availability
92      const conflictingBookings = await Booking.find({
93          equipment: equipment,
94          $or: [
95              { $and: [{ startTime: { $lt: end } }, { endTime: { $gt: start } }] }
96          ]
97      });
98
99      const bookedQuantity = conflictingBookings.reduce((total, booking) => total + booking.quantity, 0);
100     const availableQuantity = equipmentSelected.quantity - bookedQuantity;
101
102     if (quantity > availableQuantity) {
103         return res.status(400).send({ error: 'Requested quantity exceeds available quantity.' });
104     }
105
106     // Create the booking
107     const newBooking = new Booking({
108         user: req.user.userId,
109         equipment: equipment,
110         startTime: start,
111         endTime: end,
112         quantity,
113         hireOption: hireOption
114     });
115
116     await newBooking.save();
117     res.status(201).send(newBooking);
118 } catch (err) {
119     res.status(500).send({ error: err.message });
120 }
```

Conditional Statement

1

Challenge Number 1

Database Operation

Reduction Loop +
Data Calculation

Conditional Statement

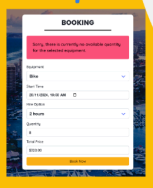
New Instance

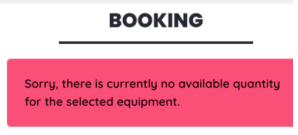
Database Operation

Error Handling

Key Challenges Faced

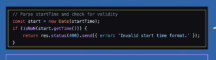
1 The complex logic that checks if the desired quantity of equipment would be available at the selected time for the selected length of hire (described previously).





2 Ensuring the selected Booking time gets displayed to the user and admin correctly.

Challenge 2 - Correctly Displaying Time



• Broke Down the Code

Challenge 2 - Correctly Displaying Time

```
// Parse startTime and check for validity
const start = new Date(startTime);
if (isNaN(start.getTime())) {
    return res.status(400).send({ error: 'Invalid start time format.' });
}
```

Initial Code

- Broke Down the Code
- Consulted MDN Web docs and remembered about Unix Epoch - number of milliseconds since January 1, 1970.
- Took into account the conversion needed for AEST

Problem Solving

```
// Subtract 10 hours from start time if in correct format
if(start) {
    start.setHours(start.getHours() - 10);
}
```

Added Code