Inspection Code - Lab Rats
For our application, we used the Next.js development framework which utilizes the App router to allow file based routing within the application. The application is built using the React framework and uses the Chakra UI component library.

This is the Cart.tsx file, which is to handle and keep track of orders (not fully implemented as of yet).

```
"use client";

import { Box, Button, Icon } from "@chakra-ui/react";
import { useState } from "react";
import { CiShoppingCart } from "react-icons/ci";
import CartDialog from "@/components/ui/cart/cartDialog";

const Cart = () => {
  const [DialogVisible, setDialogVisible] = useState(false);
  const handleToggleDialog = () => setDialogVisible(!DialogVisible);

  return (
    <Box>
      <Button
        mx={2}
        aria-label="view-cart"
        onClick={handleToggleDialog}
        bg="orange.500"
        _hover={{ bg: "orange.600" }}
      >
        <Icon>
          <CiShoppingCart size="1.5em" />
        </Icon>
      </Button>
      // Uses a nested component CartDialog for organization and state handling
      <CartDialog
        cartItems={[]}
        visible={DialogVisible}
        onClose={handleToggleDialog}
      />
    </Box>
  );
};
export default Cart;
```

This is the nested CartDialog component, which does the actual handling and what to display. If the cart is empty a dialog box will be displayed instead of the list of orders.

```
import { Box, Text, Dialog, Portal, Button, Link } from "@chakra-ui/react";

type CartDialogProps = {
  cartItems: any[];
  visible: boolean;
  onClose: () => void;
};
```

// To help with code readability, styling is defined and passed as props to Chakra UI components

```
const CartDialog = ({ cartItems, visible, onClose }: CartDialogProps) => {
  const dialogStyles = {
    maxW: "600px",
    width: "100%",
    position: "relative",
    p: 4,
    margin: "0 auto",
  };

  const exitButtonStyles = {
    position: "absolute",
    top: 4,
    right: 4,
  };

  if (!visible) return null;

  return (
    <Box>
      <Dialog.Root
        size={{ mdDown: "full", md: "lg" }}
        open={visible}
        placement="center"
      >
        <Portal>
          <Dialog.Backdrop>
            <Dialog.Positioner css={dialogStyles}>
              <Dialog.Content>
                <Dialog.Header>
                  <Dialog.Title textStyle="lg">Cart</Dialog.Title>
                  <Button onClick={onClose} css={exitButtonStyles}>
                X
              </Button>
            </Dialog.Header>
            <Dialog.Body>
              {cartItems.length == 0 ? (
                <Box textAlign="center" py={4} px={4}>
                  <Text textStyle="lg" mb={4}>
                    Looks like your cart is empty.
                  </Text>
                  <Link
                    href="/menu"
                    color="black"
                    style={{ textDecoration: "none" }}
                  >
                    <Button>Start Your Order</Button>
                  </Link>
                </Box>
              ) : (
                <Box>{/* Cart items would go here */}</Box>
              )}
            </Dialog.Body>
          </Dialog.Content>
        </Dialog.Positioner>
      </Dialog.Backdrop>
    </Portal>
  </Dialog.Root>
</Box>
);
};

export default CartDialog;
```

This is the login page to handle sign up and sign ins. (Not fully implemented yet).

```
import {
  Container,
  Flex,
  Box,
  Stack,
  Heading,
  Text,
  Field,
  Checkbox,
  Input,
  Link,
  Button,
} from "@chakra-ui/react";
import type { Metadata } from "next";

export const metadata: Metadata = {
  title: "Login",
};

const Login = () => {
  return (
    <Container maxW="lg" py={12}>
      <Flex align="center" justify="center">
        <Box
          w="full"
          p={{ base: 6, md: 8 }}
          borderWidth="1px"
          borderRadius="lg"
          boxShadow="sm"
          bg="white"
          borderColor="gray.200"
        >
          <Stack gap={6}>
            <Stack gap={1} textAlign="center">
              <Heading size="lg" color="gray.600">
                Welcome back
              </Heading>
              <Text color="gray.600">Sign in to your account</Text>
            </Stack>

            <Stack as="form" gap={4}>
              <Field.Root>
                <Field.Label htmlFor="email" color="gray.600">
    <Field.RequiredIndicator />
    Email address
  </Field.Label>
  <Input
    id="email"
    type="email"
    placeholder="you@example.com"
    bg="gray.900"
    color="white"
    borderColor="white"
    _placeholder={{ color: "whiteAlpha.700" }}
    _hover={{ borderColor: "white" }}
    _focusVisible={{ borderColor: "white", boxShadow: "none" }}
  />
</Field.Root>

<Field.Root>
  <Field.Label htmlFor="password" color="gray.600">
    <Field.RequiredIndicator />
    Password
  </Field.Label>
  <Input
    id="password"
    type="password"
    placeholder="••••••••"
    bg="gray.900"
    color="white"
    borderColor="white"
    _placeholder={{ color: "whiteAlpha.700" }}
    _hover={{ borderColor: "white" }}
    _focusVisible={{ borderColor: "white", boxShadow: "none" }}
  />
</Field.Root>

<Flex justify="space-between" align="center">
  <Checkbox.Root color="gray.600">
    <Checkbox.HiddenInput id="remember" />
    <Checkbox.Control />
    <Checkbox.Label>Remember me</Checkbox.Label>
  </Checkbox.Root>
  <Link href="#" color="blue.500" fontWeight="semibold">
    Forgot password?
  </Link>
</Flex>
```

```
            <Button
              size="md"
              mt={4}
              type="submit"
              css={{ bg: "orange.500", _hover: { bg: "orange.600" } }}
            >
              Sign in
            </Button>
          </Stack>

          <Text fontSize="sm" color="gray.600" textAlign="center">
            Don&apos;t have an account?{" "}
            <Link href="#" color="blue.500" fontWeight="semibold">
              Sign up
            </Link>
          </Text>
        </Stack>
      </Box>
    </Flex>
  </Container>
 );
};

export default Login;
```

This is the backend code for performing requests for Cart.

```
import {
  getCartController,
  putCartController,
  clearCartController,
} from "../../../lib/domains/cart/controller.js";

function cid(req) {
  return req.headers.get("x-customer-id");
}

export async function GET(req) {
  const customerId = cid(req);
  if (!customerId)
    return new Response(
      JSON.stringify({ ok: false, msg: "Missing x-customer-id" }),
      { status: 401 }
    );
  const out = await getCartController(customerId);
  return new Response(JSON.stringify(out), {
    status: 200,
    headers: { "Content-Type": "application/json" },
  });
}

export async function PUT(req) {
  const customerId = cid(req);
  if (!customerId)
    return new Response(
      JSON.stringify({ ok: false, msg: "Missing x-customer-id" }),
      { status: 401 }
    );
  let body;
  try {
    body = await req.json();
  } catch {
    return new Response(JSON.stringify({ ok: false, msg: "Invalid JSON" }), {
      status: 400,
    });
  }
  const { items = [], updatedAt: clientUpdatedAt } = body || {};
  try {
    const out = await putCartController(customerId, items, clientUpdatedAt);
    return new Response(JSON.stringify(out), {
      status: out.status || 200,
      headers: { "Content-Type": "application/json" },
    });
  } catch (e) {
    return new Response(JSON.stringify({ ok: false, msg: e.message }), {
      status: 400,
    });
  }
}

export async function DELETE(req) {
  const customerId = cid(req);
  if (!customerId)
    return new Response(
      JSON.stringify({ ok: false, msg: "Missing x-customer-id" }),
      { status: 401 }
    );
  const out = await clearCartController(customerId);
  return new Response(JSON.stringify(out), {
    status: 200,
    headers: { "Content-Type": "application/json" },
  });
}
```

Backend code for Creating new user
USER Schema - Base model for user it oversees both employee and customers

```
import mongoose from "mongoose";
import bcrypt from "bcryptjs";

const { Schema } = mongoose;

const UserBaseSchema = new Schema(
  {
    firstName: { type: String, required: true, trim: true },
    lastName: { type: String, required: true, trim: true },
    email: {
      type: String,
      required: true,
      unique: true,
      lowercase: true,
      trim: true,
    },
    password: { type: String, required: true, minlength: 6, select: false }, // hashed
  },
  {
    timestamps: true,
    discriminatorKey: "role",
    toJSON: {
      virtuals: true,
      transform: (_doc, ret) => {
        delete ret.password;
        delete ret.__v;
        return ret;
      },
    },
  }
);

UserBaseSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

UserBaseSchema.methods.comparePassword = function (candidate) {
  return bcrypt.compare(candidate, this.password);
};
```

```
const User = mongoose.models.User || mongoose.model("User", UserBaseSchema);
export default User;
```

Employee Schema - Built upon the user schema adds objects needed for an employee

```
import mongoose from "mongoose";
import User from "../../user/schema/user.schema.js";

const { Schema } = mongoose;

const EmployeeSchema = new Schema({
  roleTitle: { type: String, enum: ["owner", "manager", "chef"], required: true },
  hireDate:  { type: Date, default: Date.now },
  isActive:  { type: Boolean, default: true },
});

const Employee = User.discriminator("employee", EmployeeSchema);
export default Employee;
```

Employee Controller - Handles employee-related operations like validating employee data, registering staff members with specific roles, and managing their employment details.

```
import { connectDB } from "../../database/connect.js";
import Employee from "./schema/employee.schema.js";
import User from "../user/schema/user.schema.js";
import bcrypt from "bcryptjs";
//import { signToken } from "../user/auth.js";

function sanitizeUser(doc) {
  if (!doc) return null;
  const obj = doc.toObject ? doc.toObject() : doc;
  delete obj.password;
  delete obj.__v;
  return obj;
}
function validateLoginData(body) {
  const errors = [];
  if (
    !body.email ||
    typeof body.email !== "string" ||
    !/\S+@\S+\.\S+/.test(body.email)
  ) {
    errors.push({ path: "email", msg: "Valid email required" });
  }
  if (!body.password || typeof body.password !== "string") {
    errors.push({ path: "password", msg: "Password is required" });
  }
  return errors;
}
function validateEmployeeData(body) {
  const errors = [];
  if (
    !body.firstName ||
    typeof body.firstName !== "string" ||
    !body.firstName.trim()
  ) {
    errors.push({ path: "firstName", msg: "firstName is required" });
  }

  if (
    !body.lastName ||
    typeof body.lastName !== "string" ||
    !body.lastName.trim()
  ) {
    errors.push({ path: "lastName", msg: "lastName is required" });
  }

  if (
    !body.email ||
    typeof body.email !== "string" ||
    !/\S+@\S+\.\S+/.test(body.email)
  ) {
    errors.push({ path: "email", msg: "Valid email required" });
  }

  if (
    !body.password ||
    typeof body.password !== "string" ||
    body.password.length < 6
  ) {
    errors.push({
      path: "password",
      msg: "Password must be at least 6 characters",
    });
  }
  const allowedRoles = ["owner", "manager", "chef"];
  if (!body.roleTitle || !allowedRoles.includes(body.roleTitle)) {
    errors.push({
      path: "roleTitle",
      msg: 'roleTitle must be one of: "owner", "manager", "chef"',
    });
  }

  if (body.isActive !== undefined && typeof body.isActive !== "boolean") {
    errors.push({ path: "isActive", msg: "isActive must be a boolean" });
  }

  if (body.hireDate !== undefined && Number.isNaN(Date.parse(body.hireDate))) {
    errors.push({
      path: "hireDate",
      msg: "hireDate must be a valid date string",
    });
  }

  return errors;
}

export async function signupEmployeeController(bodyRaw) {
  await connectDB();

  const body = {
    ...bodyRaw,
    email:
      typeof bodyRaw.email === "string"
        ? bodyRaw.email.toLowerCase().trim()
        : bodyRaw.email,
  };

  const validationErrors = validateEmployeeData(body);
  if (validationErrors.length > 0) {
    return { status: 400, body: { ok: false, errors: validationErrors } };
  }

  const {
    firstName,
    lastName,
    email,
    password,
    roleTitle,
    hireDate,
    isActive,
  } = body;

  const existing = await User.findOne({ email }).lean();
  if (existing) {
    return { status: 409, body: { ok: false, msg: "Email already in use" } };
  }

  try {
    const newEmployee = await Employee.create({
      firstName,
      lastName,
      email,
      password,
      roleTitle,
      hireDate: hireDate ? new Date(hireDate) : undefined,
      isActive,
    });

    // // Sign JWT
    // const token = signToken({
    //   sub: newEmployee._id.toString(),
    //   role: "employee",
    //   roleTitle: newEmployee.roleTitle,
    // });

    return {
      status: 201,
      body: {
        ok: true,
        msg: "Employee registered successfully",
        data: newEmployee.toJSON(),
      },
    };
  } catch (err) {
    if (err?.code === 11000) {
      return { status: 409, body: { ok: false, msg: "Email already in use" } };
    }
    console.error("signupEmployeeController error:", err);
    return { status: 500, body: { ok: false, msg: "Server error" } };
  }
}
export async function loginEmployeeController(bodyRaw) {
  await connectDB();
```

```
const body = {
  ...bodyRaw,
  email:
    typeof bodyRaw.email === "string"
      ? bodyRaw.email.toLowerCase().trim()
      : bodyRaw.email,
};

const validationErrors = validateLoginData(body);
if (validationErrors.length > 0) {
  return { status: 400, body: { ok: false, errors: validationErrors } };
}

const { email, password } = body;

try {
  const employee = await Employee.findOne({ email }).select("+password");
  if (!employee) {
    return { status: 404, body: { ok: false, msg: "User not found" } };
  }

  if (typeof employee.password !== "string") {
    return {
      status: 400,
      body: { ok: false, msg: "Account has no password set" },
    };
  }

  const isMatch = await bcrypt.compare(password, employee.password);
  if (!isMatch) {
    return { status: 401, body: { ok: false, msg: "Invalid credentials" } };
  }

  // const token = signToken({
  //   sub: employee._id.toString(),
  //   role: "employee",
  //   // roleTitle: employee.roleTitle
  // });

  return {
    status: 200,
    body: {
      ok: true,
      msg: "Login successful",
        data: sanitizeUser(employee),
        //token,
      },
    };
  } catch (err) {
    console.error("loginEmployeeController error:", err);
    return { status: 500, body: { ok: false, msg: "Server error" } };
  }
}
```

Customer Schema - Built on the base of user schema with object needed for customer

```
import mongoose from "mongoose";
import User from "../../user/schema/user.schema.js";

const { Schema } = mongoose;

const AddressSchema = new Schema(
  {
    street: { type: String },
    city: { type: String },
    state: { type: String },
    zip: { type: String },
  },
  { _id: false }
);

const CustomerSchema = new Schema({
  phoneNumber: { type: String, required: true, trim: true },
  address: { type: AddressSchema, required: true },
  orderHistory: [{ type: Schema.Types.ObjectId, ref: "Order" }],
});

const Customer = User.discriminator("customer", CustomerSchema);
export default Customer;
```