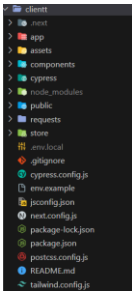
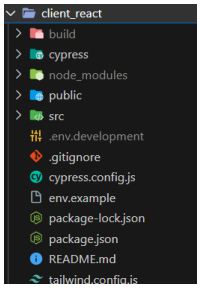
Objectives

All Objectives are completed. Following are images of completed objectives.

Tech Stack:

- ✓ Frontend: Next.js **(Completed using both Next.js and React.js)**





- ✓ Design: Tailwind CSS, Bootstrap, or you can use simple CSS **(Completed using Tailwind CSS and simple CSS)**

- ✓ Backend: Express.js **(Completed)**

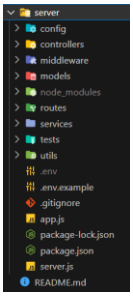
```
import express from "express";
import bodyParser from "body-parser";
import cookieParser from "cookie-parser";
import cors from "cors";
import errorMiddleware from "./middleware/error.js";
import todoRoutes from "./routes/todoRoute.js";
import userRoutes from "./routes/userRoute.js";

const app = express(); // create app

// use packages
app.use(express.json());
app.use(cookieParser());
app.use(
    cors({
        origin: true,
        credentials: true,
        exposedHeaders: ["Set-Cookie", "Date", "ETag"],
    })
);
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/api/v1/todo", todoRoutes); // todo routes
app.use("/api/v1/user", userRoutes); // user routes

app.use(errorMiddleware); // error middleware

export default app;
```

- ✓ Database: Mongo DB **(Completed using MongoDB and Mongoose)**

```
import mongoose from "mongoose";

const todoSchema = new mongoose.Schema(
    {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            required: [true, "Please enter user id"],
        },
        todo: {
            type: String,
            required: [true, "Please enter todo item
            name"],
            maxLength: [20, "Todo item name cannot exceed
            20 characters"],
        },
        completed: {
            type: Boolean,
            default: false,
        },
        completionTime: {
            type: Date,
            default: null,
        },
    },
    { timestamps: true }
);

var Todo = mongoose.model("Todo", todoSchema);
export default Todo;
```

API Implementation:

- ✓ If you choose to use REST API:
  - ✓ Follow REST principles strictly when designing the API endpoints. **(Completed by following REST Principles)**

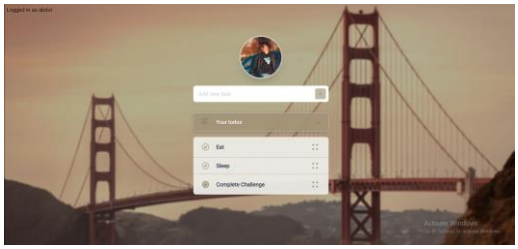


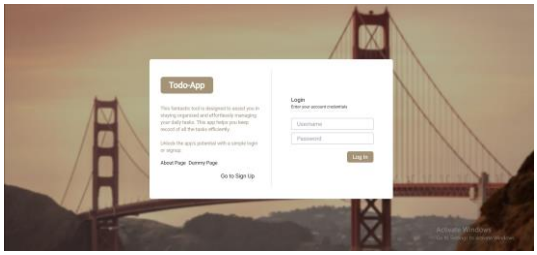
Requirements:

1. Design and Layout:

   - ✓ Replicate the provided design example for the todo list app. **(Completed and images are provided)**

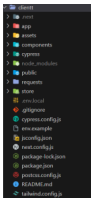
   - ✓ Use the chosen design framework to create a visually appealing and responsive user interface. **(Completed using Next.js and React.js)**

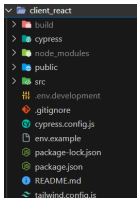




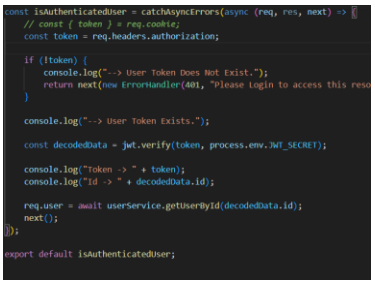
2. Frontend Development:

   - ✓ Use above mentioned technologies to develop the frontend of the application. **(Completed the frontend as asked)**



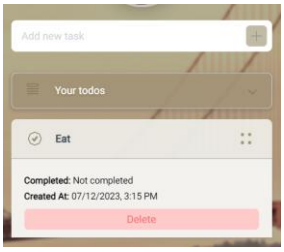


- ✓ Add authentication flow (login, register). You can use any design for register and login routes. **(Completed Login/Signup Functionality)**
  Note: Bonus Task (Token revalidation) **(Completed using JWT)**

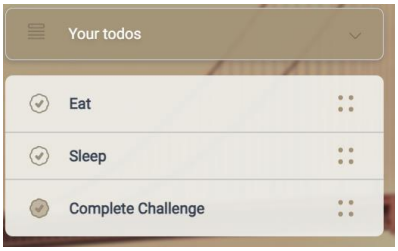


```
const isAuthenticatedUser = catchAsyncErrors(async (req, res, next) => {
    // const { token } = req.cookie;
    const token = req.headers.authorization;

    if (!token) {
        console.log("--> User Token Does Not Exist.");
        return next(new ErrorHandler(401, "Please Login to access this reso
    }

    console.log("--> User Token Exists.");

    const decodedData = jwt.verify(token, process.env.JWT_SECRET);

    console.log("Token -> " + token);
    console.log("Id -> " + decodedData.id);

    req.user = await userService.getUserById(decodedData.id);
    next();
});

export default isAuthenticatedUser;
```

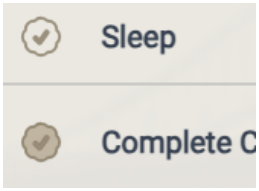
- ✓ Implement features for adding, editing, and deleting tasks. **(Completed Features)**
  Note: TODO route should be private, cannot be accessible without authentication. **(Completed by making main page private)**



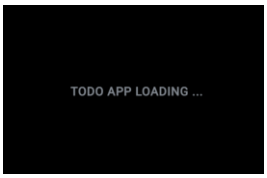
- ✓ Display the list of tasks with relevant details. **(Completed)**

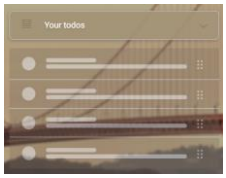


- ✓ Include options to mark tasks as completed or pending. **(Completed)**



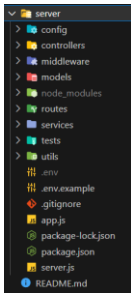
- ✓ Apply appropriate styling and animations as per the design. **(Completed)**





3. Backend Development:

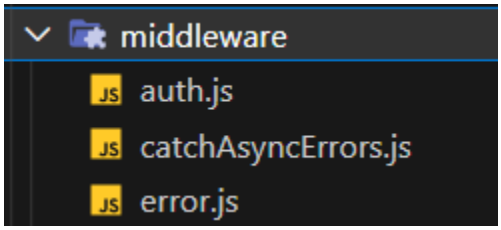
   ✓ Use above mentioned technologies to build the backend of the application. **(Completed the backend as asked)**



   ✓ Create RESTful API endpoints for CRUD operations related to tasks. **(Completed REST Api)**

```
// main url for the backend server
const mainUrl = `${process.env.REACT_APP_BACKEND_API_URL}/api/v1/todo`;

// get all to do items
export const getCompleteList = async () => await axios.get(`${mainUrl}/all`);

// add new item
export const addNewItem = async (data) =>
    await axios.post(`${mainUrl}/add`, data);

// update an item
export const updateItem = async (id, data) =>
    await axios.put(`${mainUrl}/update/${id}`, data);

// delete an item
export const deleteItem = async (id) =>
    await axios.delete(`${mainUrl}/delete/${id}`);
```

   ✓ Implement necessary validation and error handling for the API. **(Completed using Auth and Error Middlewares)**



4. Database Integration:

   ✓ Set up the database and establish a connection with the backend. **(Completed using mongodb cloud)**

```
import mongoose from "mongoose";
mongoose.set("strictQuery", true);

const connectDB = (dbPath) => {
    mongoose
        .connect(dbPath, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        })
        .then((data) => {
            console.log(
                `-> Mongodb connected with server: ${data.
                connection.host}\n`
            );
        });
};

export default connectDB;
```

- ✓ Design the database collections to store task-related information. **(Completed using Mongoose Schema)**

```
import mongoose from "mongoose";

const todoSchema = new mongoose.Schema(
    {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            required: [true, "Please enter user id"],
        },
        todo: {
            type: String,
            required: [true, "Please enter todo item
            name"],
            maxLength: [20, "Todo item name cannot exceed
            20 characters"],
        },
        completed: {
            type: Boolean,
            default: false,
        },
        completionTime: {
            type: Date,
            default: null,
        },
    },
    { timestamps: true }
);

var Todo = mongoose.model("Todo", todoSchema);
export default Todo;
```

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

const userSchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: [true, "Please enter your Name"],
            uniue: [true, "Username already in use"],
            maxLength: [20, "Name cannot exceed 20 characters"],
            minLength: [3, "Name should have more than 3 characters"],
        },
        password: {
            type: String,
            required: [true, "Please enter your Password"],
            minLength: [8, "Password should have more than 8 characters"],
            // select: false,
        },
    },
    { timestamps: true }
);
```

- ✓ Implement necessary queries or operations to interact with the database. **(Completed in service layer)**

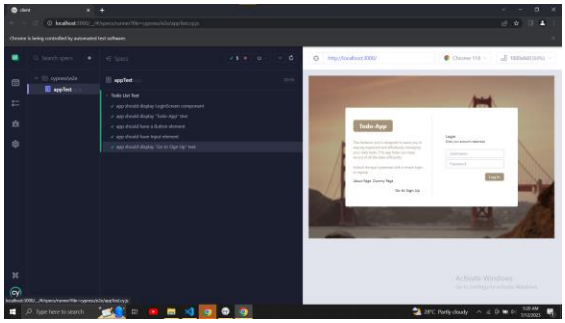
```
import User from "../models/userModel.js";

// GET USER BY NAME
const getUserByName = async (name) => {
    return await User.findOne({ name: name });
};

// GET USER BY ID
const getUserById = async (userId) => {
    return await User.findById(userId);
};

// CREATE NEW USER
const createUser = async (name, password) => {
    return await User.create({
        name: name,
        password: password,
    });
};

export default { getUserById, getUserByName, createUser };
```

```
import mongoose from "mongoose";
import Todo from "../models/todoModel.js";

// GET ITEM BY ID
const getItemById = async (itemId) => {
    return await Todo.findById(itemId);
};

// ALL ITEMS
const getAllItems = async (userId) => {
    return await Todo.aggregate([
        {
            $match: {
                user: mongoose.Types.ObjectId(userId),
            },
        },
        {
            $sort: {
                createdAt: -1,
            },
        },
    ]);
};

// ADD ITEM
const addItem = async (todo, userId) => {...
};
```

5. Testing:

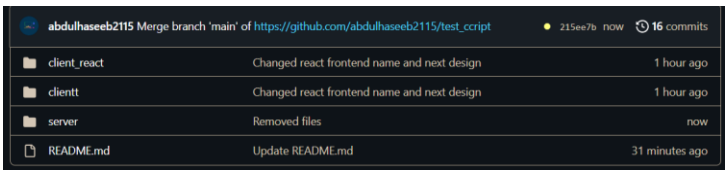
- ✓ Test the frontend and backend functionalities to ensure proper working. **(Completed using manual testing and using basic features of Jest and Cypress)**

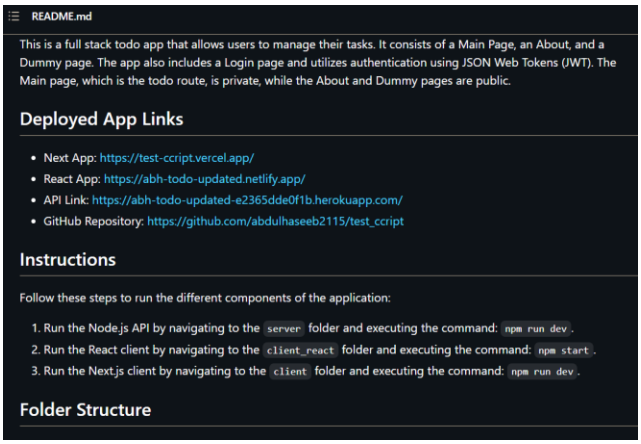
```
PASS tests/app.test.js (7.364 s)
  Login
    √ should login to todo app (4304 ms)
  Add New Todo Item
    o skipped should add a new todo item
  Delete An Item
    o skipped should delete a todo item
  Update Task Controller
    o skipped should update a task

Test Suites: 1 passed, 1 total
Tests:       3 skipped, 1 passed, 4 total
Snapshots:   0 total
Time:        7.592 s, estimated 8 s
Ran all test suites.
```



6. Documentation and Deployment:

   ✓ Push your code to Github or Gitlab and provide clear and add Readme.md file to provide concise documentation on how to set up and run the application. **(Completed using Github)**





   ✓ Deploy the application on a hosting platform of your choice (e.g., Heroku, Vercel, Netlify) and provide the deployment URL. **(Completed using Vercel, Netlift and Heroku)**

   Next App: https://test-ccript.vercel.app/

   React App: https://abh-todo-updated.netlify.app/

   API Link: https://abh-todo-updated-e2365dde0f1b.herokuapp.com/

   GitHub Repository: https://github.com/abdulhaseeb2115/test_ccript

END