University of Technology, Jamaica

Advanced Programming (CIT3009)

Group Assignment

Jordan Liu (1506757)

Tarique Jemison (1703228)

Racquel Bailey (1701406)

Gilroy Gordon

Table of Contents

Selected Tools Used in the Project:

While completing this project, several tools were used. These tools enabled the group to concretize concepts and turn the blueprint into reality. The tools utilized were the MERN stack, Visual Studio Code, Git, Gitlab and Docker. It should be noted that the latest stable versions of all the listed tools were utilized.

The MERN stack comprises 4 key web development technologies which are MongoDB, Express, React, Node.js. (Cmarix, 2019)

MongoDB is a general-purpose, document-based, distributed database built for modern application developers and for the cloud era. It is a NoSQL class-platform, which means it is a non-traditional database that provides a mechanism for storage and retrieval of data that is modeled rather than the traditional tabular relation used in relational databases.

Express.js is an open-source web application framework that works as a framework for Node.js. It provides mechanisms such as writing handlers for requests, integrates with rendering engines to generate responses. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

React is a JavaScript-based front-end library built and maintained by Facebook which is used for UI development. it is a declarative, flexible and efficient javascript library used for building user interfaces,

According to Cmarix (2019), Node.js is the JavaScript framework for server-side or backend development. node is an open-source, cross-platform runtime environment that allows developers to create server-side tools and applications in javascript. It has a mirage of benefits including a node package manager, that allows access for thousands of reusable packages.

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications. (Visual Studio Code, 2016). Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging. It also has git commands built-in, is extensible and customizable.

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. (Git, n.d). It is currently the most widely used control system. It was designed with performance, flexibility, and security in mind to make lives easier for current and future potential developers. GitLab is a complete DevOps platform, delivered as a single application. It is a web-based DevOps lifecycle tool providing git-repository manager, issue tracking, CI/CD pipeline features and is also an open-source and core project.

According to opensource.com (n.d), a docker is a tool designed to make it easier to create, deploy, and run applications by using containers. it enables developers to easily ship, pack and run any application as a lightweight, portable and self-sufficient container which is able to virtually run anywhere. It should be noted that the containers are able to do this by enabling developers to isolate code into a single container, which makes it easier to modify and update the program.

.

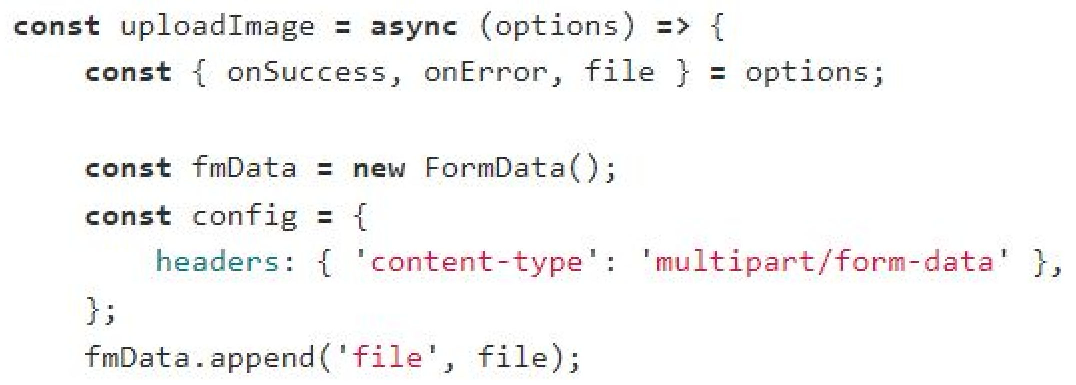
Research Concepts

Use of the S.O.L.I.D principles

SOLID is an acronym for five of the software design principles that promote flexibility and maintainability. These principles, when combined, make it easy for developers to build software programs that are easily maintained and extended. It also makes it possible for developers to easily refactor code and is a part of agile software development. The acronyms stand for Single-Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle.

Single Responsibility Principle

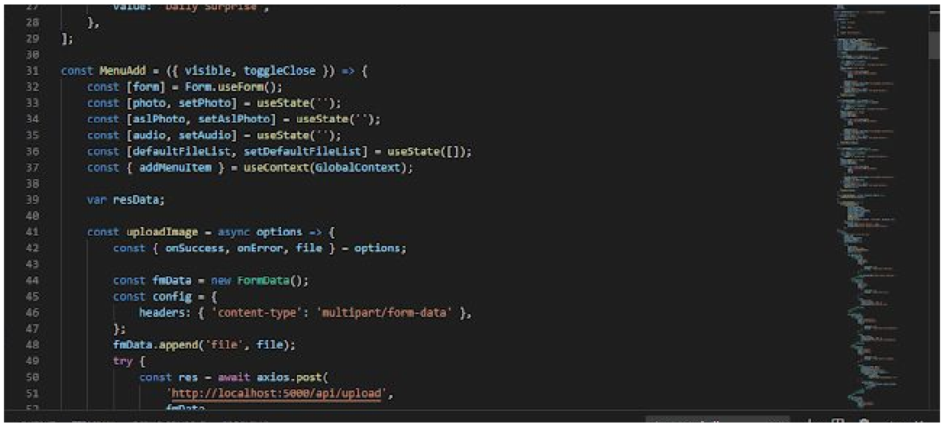
This principle states that an entity should only have one responsibility. It also should have one and only one reason to change. The changes to this is less likely to affect other concerns and it allows for separate concerns. An example of this principle used in this project is in the module uploadImage where the sole responsibility is uploading an image.



```
const uploadImage = async (options) => {
    const { onSuccess, onError, file } = options;

    const fmData = new FormData();
    const config = {
        headers: { 'content-type': 'multipart/form-data' },
    };
    fmData.append('file', file);
```

Open/Closed Principle

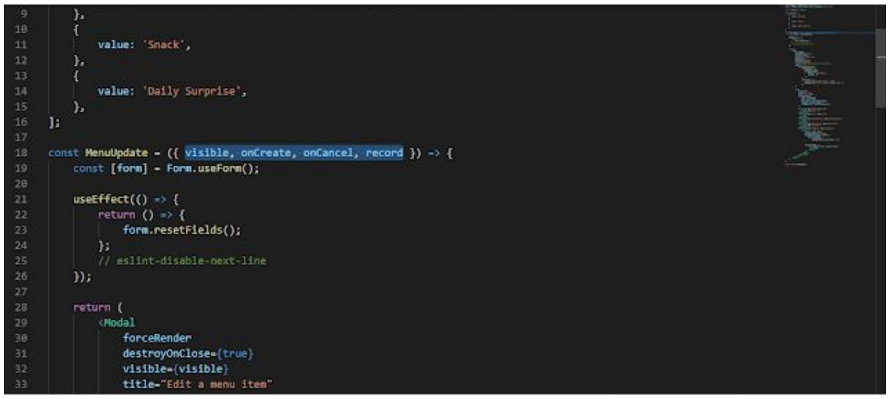
This principle dictates that objects or entities should be open for extension but closed for modification. Meaning, a class should be easily extendable without modification of the class itself and be able to inherit to add functionality.



```
            value: 'Daily Surprise',
        },
];

const MenuAdd = ({ visible, toggleClose }) => {
    const [form] = Form.useForm();
    const [photo, setPhoto] = useState('');
    const [aslPhoto, setAslPhoto] = useState('');
    const [audio, setAudio] = useState('');
    const [defaultFileList, setDefaultFileList] = useState([]);
    const { addMenuItem } = useContext(GlobalContext);

    var resData;

    const uploadImage = async options => {
        const { onSuccess, onError, file } = options;

        const fmData = new FormData();
        const config = {
            headers: { 'content-type': 'multipart/form-data' },
        };
        fmData.append('file', file);
        try {
            const res = await axios.post(
                'http://localhost:5000/api/upload',
                fmData
```

Liskov Substitution Principle

This principle states that the objects in the program should be replaceable without altering the program’s correctness. Meaning, every sub or derived class should be substitutable for their parent or base class. Liskov substitution is achieved when abstract types are being utilized as parameters.



```
    },
    {
        value: 'Snack',
    },
    {
        value: 'Daily Surprise',
    },
];

const MenuUpdate = ({ visible, onCreate, onCancel, record }) => {
    const [form] = Form.useForm();

    useEffect(() => {
        return () => {
            form.resetFields();
        };
        // eslint-disable-next-line
    });

    return (
        <Modal
            forceRender
            destroyOnClose={true}
            visible={visible}
            title="Edit a menu item"
```

Interface Segregation Principle

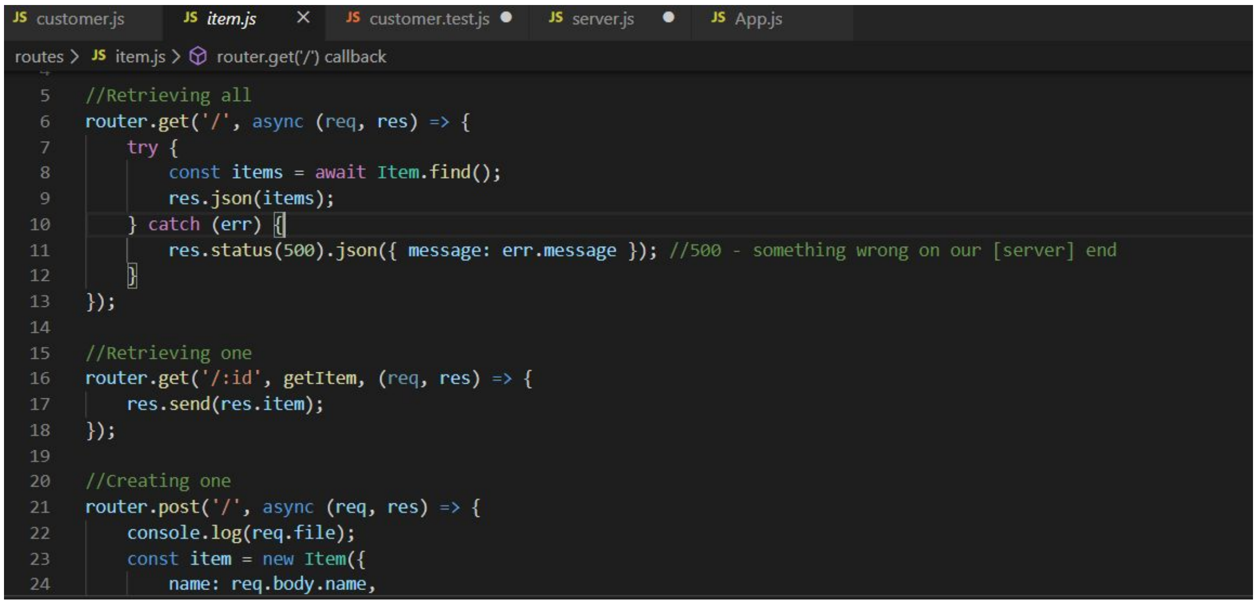
The interface segregation principle highlights that a client should never be forced to implement an interface that it doesn't use or clients should not be forced to depend on methods they do not use. (Oloruntoba, S., 2015). It separates behaviour and prevents the child classes from implementing unrelated behaviour. Because of the framework selected, interfaces were not explicitly used in the way defined and used in the principle.

Dependency Inversion Principle

Dependency Inversion entails that entities must depend on abstractions and not concretions. It states that high-level modules must never depend on low-level modules but instead, depend on abstractions. This principle also allows for decoupling.

The Repository Pattern

Repositories are classes or components that encapsulate the logic required to access data sources. The pattern reduces redundancy of code, fewer potential errors and forces the programmer to work using the same pattern. It utilizes abstracting to reduce coupling and it is the container where the data access logic is stored. An instance used in this project is in the routes folder, where the API requests are found, specifically in items.js. It shows a get request to retrieve all and one item. Since the repository pattern deals with the logic in accessing data, the item.js file shows the practicality of the pattern as shown below.



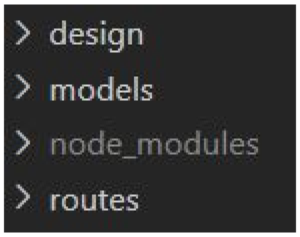
```
//Retrieving all
router.get('/', async (req, res) => {
    try {
        const items = await Item.find();
        res.json(items);
    } catch (err) {
        res.status(500).json({ message: err.message }); //500 - something wrong on our [server] end
    }
});

//Retrieving one
router.get('/:id', getItem, (req, res) => {
    res.send(res.item);
});

//Creating one
router.post('/', async (req, res) => {
    console.log(req.file);
    const item = new Item({
        name: req.body.name,
```

The Model-View-Controller Pattern

The Model-View-Controller Pattern, also called the MVC Pattern, states that an application must consist of a data model, presentation information and control information with each of these separated into different objects. It is seen as an architectural pattern as it mostly relates to the User Interface layer of an application. The model consists of application data and consists of no logic on how to present the data to the user. The view presents this data to the user and also knows how to access the model’s data but does not know what the data means. The controller is the middleware between the model and the view. It executes reactions to the events. This is seen throughout the project in the model and routes folders which would represent the model and control layers.

The Singleton Pattern

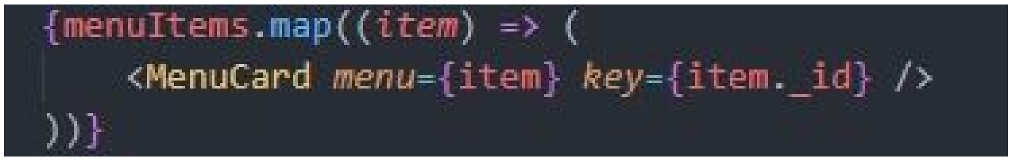
Maintains one instance of a resource to be utilized by many objects. Instance used in project:



```
> design
> models
> node_modules
> routes
```

The Factory Pattern

The creational pattern that uses factory methods to deal with the problem of creating objects without explicitly specifying the class of the object that will be created. This is seen in the snippet below in the Menu.js file and uses MenuCard.js component.



```
{menuItems.map((item) => (
    <MenuCard menu={item} key={item._id} />
))}
```

Code Documentation Generation Tool

Code Documentation Generation tools generate documentation from source code provided. This is used with JSDoc, a JavaScript documentation generation tool.

Code Scaffolding Tool

Create-React-App (CRA) was utilized to initialize the front-end framework using React.

Source Control Management Tool

Source control is the practice of tracking and managing the changes to the code. It provides a running history of code development and helps to resolve conflicts when merging contributions. Git is the open-source code management tool that was utilized in the project to keep track of the changes made in the code

Package Management Tool

A package manager is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner. They are designed to eliminate the need for manual installs and updates. NPM was the package manager utilized, which is a JavaScript package manager.

Unit Testing and Test Automation

Unit testing is a type of software testing where individual units or components of a software are tested (Guru99, 2020). Its purpose is to validate each unit of the software code to ensure it performs as expected. Jest was utilized to create unit tests for the frontend, test automation was carried out using Gitlab CI.

Continuous Integration, Continuous Integration Server and Continuous Deployment

Continuous Integration is a development practice where developers integrate code in a shared repository frequently; in our case, the shared repository being gitlab. Integration is done on the User Acceptance Testing (UAT) branch. User acceptance testing (UAT) is the last phase of the software testing process. During UAT, actual software users test the software to make sure it can handle required tasks in real-world scenarios, according to specifications.Continuous deployment is a software development strategy for releases where any commited code that passes the automated testing phase is released in the production environment. It’s usage can be noted in the resource below. The project used a docker container along with a Gitlab runner connected to an Ubuntu server.

Pipeline: https://gitlab.com/jordanxliu/cups/pipelines

Deployed URL: http://138.197.77.172:8080/

References

A. (n.d.). axios/axios. Retrieved March 23, 2020, from https://github.com/axios/axios

Ant Design of React - Ant Design. (n.d.). Retrieved March 23, 2020, from https://ant.design/docs/react/introduce

CMARIX TechnoLabs Pvt. Ltd. (2019, September 17). Why Choose MEAN Stack and Why Choose MERN Stack for Front-End Development? Retrieved from https://www.cmarix.com/what-to-choose-mean-stack-or-mern-stack-for-front-end-development/

Equivalent of SOLID principles for functional programming. (2012, September 19). Retrieved from https://softwareengineering.stackexchange.com/questions/165356/equivalent-of-solid-principles-for-functional-programming

Express 4.x - API Reference. (n.d.). Retrieved March 23, 2020, from https://expressjs.com/en/api.html

Express/Node introduction. (2020, March 12). Retrieved from https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Git. (n.d.). Retrieved March 23, 2020, from https://git-scm.com/

MongoDB Documentation. (n.d.). Retrieved March 23, 2020, from https://docs.mongodb.com/

Oloruntoba, S. (2015, March 18). S.O.L.I.D: The First 5 Principles of Object Oriented Design. Retrieved from https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Tzur, D. (2017, January 27). 5 Principles that will make you a SOLID JavaScript Developer. Retrieved from https://thefullstack.xyz/solid-javascript

Unit Testing Tutorial: What is, Types, Tools, EXAMPLE. (2020, February 14). Retrieved from https://www.guru99.com/unit-testing-guide.html

Visual Studio Code - Code Editing. Redefined. (2016, April 14). Retrieved from https://code.visualstudio.com/

What is Docker? (n.d.). Retrieved March 23, 2020, from https://opensource.com/resources/what-docker

Why Visual Studio Code? (2016, April 14). Retrieved from https://code.visualstudio.com/docs/editor/whyvscode