Nelson Marlborough Institute of Technology
Te Whare Wānanga o Te Tau Ihu o Te Waka a Maui

WEB701
Assessment 2
Evaluate

Kenneth-John Williams-Stockdale
NMIT Student ID: 13477293
National Student Number: 133006701

1. Web frameworks

Purpose of web frameworks

Frameworks are in this case software that was developed to simplify the web development process. This makes it easier to build websites/web applications. Frameworks have many capabilities that allow the presentation of information from within the browser, environments for scripting purposes, information flows and access for underlying data/resources. These usually come in the form of application programming interfaces or better known as APIs. (What Is a Web Framework?, n.d.).

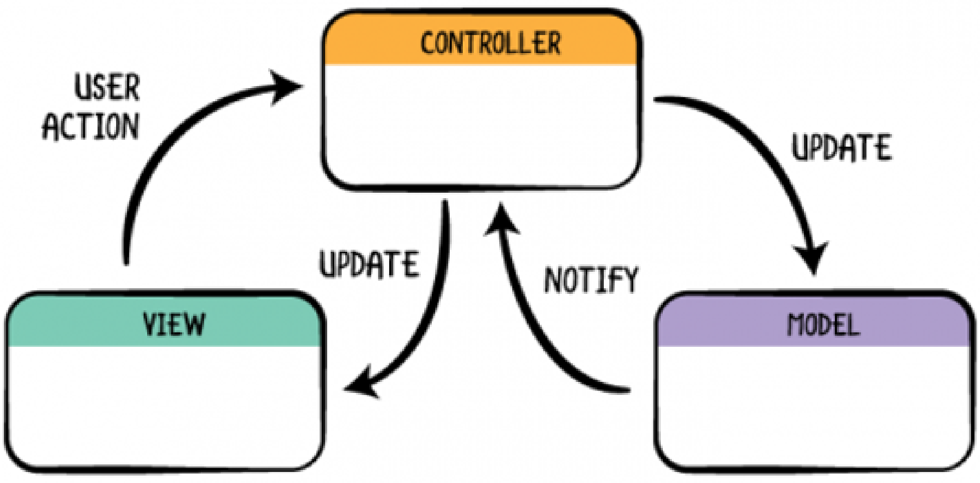
Advantages and disadvantages

One of the biggest advantages for web developers when using a web framework is that usually the frameworks are open source, have good levels of support/ security, efficient and includes many great features. (What Is a Web Framework?, n.d.). Open-source frameworks can be extremely cost effective for both developer and client. Efficiency is another quality advantage because it eliminates the need to write repetitive code, thus allowing the building of the web application to be quicker. A big disadvantage with using web frameworks is limitations. This is evident when it comes to making changes to the framework. Another factor can be that the developer may find it difficult to learn the language behind the framework, which can slow down the development process.

Common web framework features

Architecture

Many web frameworks depend on a similar architecture known as Model-View-Controller (MVC). This architecture is common because of its preferred design. It separates the logic from the app interface and forms three different parts. (Intelegain, 2019).



Model

This layer contains the business logic, guidelines, and functions. Upon getting the user input data from the controller the model layer tells/notifies how the updated interface should be displayed directly to the view.

View

The graphical layer that is a visual representation. It is the application or websites front end in this case. The view gets the user input and communicates to the controller which examines it then updates and reconstructs itself as to the directive of the model's instructions or the controller's depending on the requirement.

Controller

This layer is a translation layer. It takes input data commands and works as a middleman between both model and view layers. It notifies both layers of any changes required. For the current web assessment, I am undertaking. The chosen architecture is the same as the MVC mentioned above and makes use of the mentioned features.

Security

Input Validation

This is the process of making sure that any user input data is acceptable and consistent with the expectations of what the application allows. Any inputs that fall outside of this can produce unexpected results, bugs and or security exploits. Input validation is effective for inputs that can be confined to certain lengths or set of data. For example, when it comes to password length or username length etc. Same goes for limiting/defining options for a checkbox item. This is commonly known as positive validation or whitelisting. (Cairns & Somerfield, 2017).

In this current assessment/prototype I've used input validation on both login and signup forms. To make sure there is no data that falls outside of unexpected results but more importantly to make sure I capture the correct data needed to be used and showcased.

Hash Passwords

Protecting sensitive information is a must to protect assets from attackers and users from themselves. It's important not to store passwords and user credentials as they are into the database. To combat this, storing a hashed password with a hashing algorithm will allow for a one-way transformation from input to output from which the initial input is nearly impossible to recover. Using variations of the hash or salt in this case allows a same variation of password to have a different hash value so that no two values of password are the same.

The most widely used algorithms are scrypt and bcrypt. For this assessment I will be using bcrypt for protecting user's password and sensitive credentials.

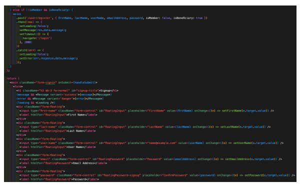
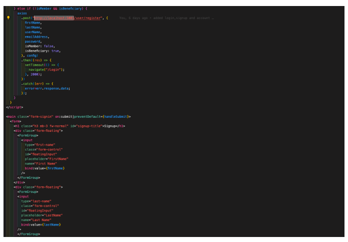
Responsive design

Most frameworks offer a responsive framework that help developers create websites that are accessible, intuitive, and able to be viewed across different browsers and different devices. This is accomplished by utilising classes that can be used and formatted to many devices and browsers that are being used.

What this means is that in the current age of development many users use mobile platforms. With these frameworks the layouts of webpages can be changed as needed so certain elements will be rendered or disappear depending on device and screen size. (An, 2021).

Responsive design is a requirement for this prototype/ assessment. It will be included within the design and prototype stages. This will allow the prototype site to be mobile friendly and responsive for viewing on any device.

2.1 Website requirement proof of concept

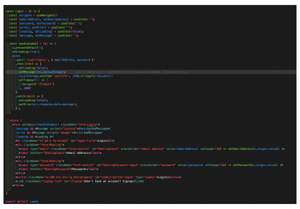
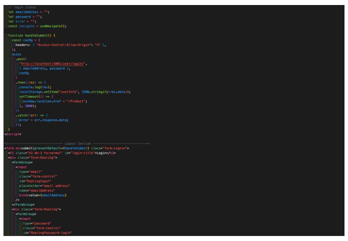
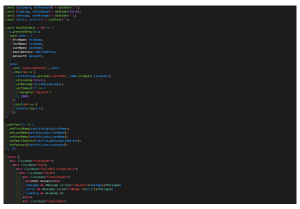
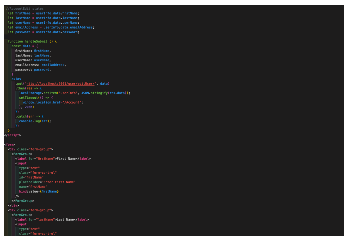
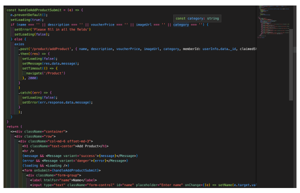
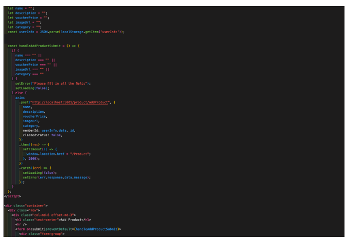
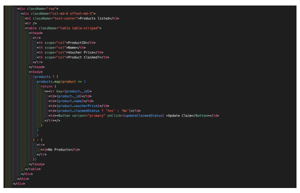
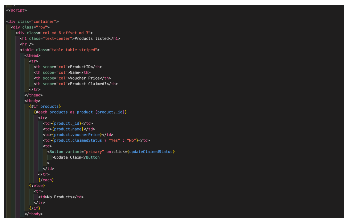
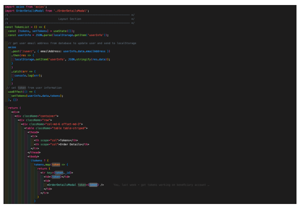
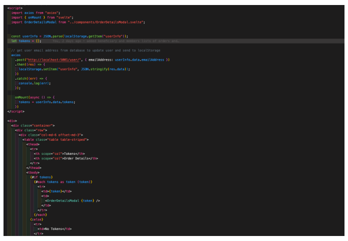
Requirements	MERN Stack	MESN Stack	Differences
1. Charity members and beneficiaries can register	 (signup.js)	 (signup.svelte)	There is few subtle differences used between these two snippets. The most noticeable is in the HTML where the MESN stack does not use onchange event handlers but instead uses the bind method.

```
    } else if (!isMember && isBeneficiary) {
      axios
        .post('/user/register', { firstName, lastName, userName, emailAddress, password, isMember: false, isBeneficiary: true })
        .then((res) => {
          setLoading(false);
          setMessage(res.data.message);
          setTimeout(() => {
            navigate('/login')
          }, 2000)
        })
        .catch((err) => {
          setLoading(false);
          setError(err.response.data.message);
        });
    }
  };

  return (
    <main className="form-signin" onSubmit={handleSubmit}>
      <form>
        <h1 className="h3 mb-3 fw-normal" id="signup-title">Signup</h1>
        {message && <Message variant='success'>{message}</Message>}
        {error && <Message variant='danger'>{error}</Message>}
        {loading && <Loading />}
        <div className="form-floating">
          <input type="first-name" className="form-control" id="floatingInput" placeholder="FirstName" value={firstName} onChange={(e) => setFirstName(e.target.value)} />
          <label htmlFor="floatingInput">First Name</label>
        </div>
        <div className="form-floating">
          <input type="last-name" className="form-control" id="floatingInput" placeholder="LastName" value={lastName} onChange={(e) => setLastName(e.target.value)} />
          <label htmlFor="floatingInput">Last Name</label>
        </div>
        <div className="form-floating">
          <input type="user-name" className="form-control" id="floatingInput" placeholder="name@example.com" value={userName} onChange={(e) => setUserName(e.target.value)} />
          <label htmlFor="floatingInput">User Name</label>
        </div>
        <div className="form-floating">
          <input type="email" className="form-control" id="floatingPassword" placeholder="Password" value={emailAddress} onChange={(e) => setEmailAddress(e.target.value)} />
          <label htmlFor="floatingPassword">Email Address</label>
        </div>
        <div className="form-floating">
          <input type="password" className="form-control" id="floatingPassword-signup" placeholder="ConfirmPassword" value={password} onChange={(e) => setPassword(e.target.value)} />
          <label htmlFor="floatingPassword">Password</label>
```

```
    } else if (!isMember && isBeneficiary) {
      axios
        .post("http://localhost:5001/user/register", {      You, 6 days ago • added login,signup and account …
          firstName,
          lastName,
          userName,
          emailAddress,
          password,
          isMember: false,
          isBeneficiary: true,
        }, config)
        .then((res) => {
          setTimeout(() => {
            navigate("/Login");
          }, 2000);
        })
        .catch((err) => {
          error=err.response.data;
        });
    }
  }
</script>

<main class="form-signin" on:submit|preventDefault={handleSubmit}>
  <form>
    <h1 class="h3 mb-3 fw-normal" id="signup-title">Signup</h1>
    <div class="form-floating">
      <FormGroup>
        <input
          type="first-name"
          class="form-control"
          id="floatingInput"
          placeholder="FirstName"
          name="First Name"
          bind:value={firstName}
        />
      </FormGroup>
    </div>
    <div class="form-floating">
      <FormGroup>
      <input
        type="last-name"
        class="form-control"
        id="floatingInput"
        placeholder="LastName"
        name="Last Name"
        bind:value={lastName}
      />
      </FormGroup>
```

Requirements	MERN Stack	MESN Stack	Differences
2. Log in	 (login.js)	 (login.svelte)	The same logic is used in the login form on both stacks. MESN deals with state management in a cleaner way with less code and better readability.
3. Administer their own accounts	 (AccountEdit.js)	 (AccountEdit.svelte)	In the MESN we don't need to have a useEffect as we can manage the same outcome by using state management to pull in our data and manage the presentable state of it.
4. Members can register their products	 (AddProduct.js)	 (AddProduct.svelte)	A notable change is how on submit is handled. MERN handles the `preventDefault()` method through a function where as the MESN stack uses it in the HTML form. The code snippets are very similar here but i've made a mistake when refactoring my code. I did not put a state for `setError` or `setLoading`
5. The system provides an interface that the members can use to accept a token in a transaction.	 (OrderDetailsModal.js)	 (OrderDetailsModal.svelte)	The clear difference here is the MESN each block and the MERN `.map()` method. Both of these methods loop through the variable and display the data on both users interfaces.*
6. Store and retrieve data from a server-side database (API)	 (TokenList.js)	 (TokenList.svelte)	Both MERN and MESN stacks can use Axios to store and retrieve data from a server-side database

```jsx
const Login = () => {
  const navigate = useNavigate()
  const [emailAddress, setEmailAddress] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    setLoading(true);
    axios
      .post('/user/login/', { emailAddress, password })
      .then((res) => {
        setLoading(false);
        setMessage(res.data.message);      You, 4 weeks ago • registration form to save database. Tested with va…
        localStorage.setItem('userInfo', JSON.stringify(res.data))
        setTimeout(() => {
          navigate('/Product')
        }, 2000)
      })
      .catch((err) => {
        setLoading(false);
        setError(err.response.data.message);
      });
  }

  return (
    <form onSubmit={handleSubmit} className="form-signin">
      {message && <Message variant="success">{message}</Message>}
      {error && <Message variant='danger'>{error}</Message>}
      {loading && <Loading />}
      <h1 className="h3 mb-3 fw-normal" id="login-title">Login</h1>
      <div className="form-floating">
        <input type="email" className="form-control" id="floatingInput" placeholder="email address" value={emailAddress} onChange={(e) => setEmailAddress(e.target.value)} />
        <label htmlFor="floatingInput">Email Address</label>
      </div>
      <div className="form-floating">
        <input type="password" className="form-control" id="floatingPassword-login" placeholder="password" value={password} onChange={(e) => setPassword(e.target.value)} />
        <label htmlFor="floatingPassword">Password</label>
      </div>
      <button className="w-100 btn btn-lg btn-primary" id="submit-button-login" type="submit">Login</button>
      <Link className="signup-link" to="/Signup">Don't have an account? Signup</Link>
    </form>
  )
}

export default Login
```

```svelte
  // login states
  let emailAddress = "";
  let password = "";
  let error = "";
  const navigate = useNavigate();

  function handleSubmit() {
    const config = {
      headers: { "Access-Control-Allow-Origin": "*" },
    };
    axios
      .post(
        "http://localhost:5001/user/login/",
        { emailAddress, password },
        config
      )
      .then((res) => {
        console.log(res);
        localStorage.setItem("userInfo", JSON.stringify(res.data));
        setTimeout(() => {
          window.location.href = "/Product";
        }, 2000);
      })
      .catch((err) => {
        error = err.response.data;
      });
  }
</script>

<!---------------------------- Layout Section ---------------------------->
<form on:submit|preventDefault={handleSubmit} class="form-signin">
  <h1 class="h3 mb-3 fw-normal" id="login-title">Login</h1>
  <div class="form-floating">
    <FormGroup>
      <input
        type="email"
        class="form-control"
        id="floatingInput"
        placeholder="email address"
        name="emailAddress"
        bind:value={emailAddress}
      />
    </FormGroup>
    <div class="form-floating">
      <FormGroup>
        <input
          type="password"
          class="form-control"
          id="floatingPassword-login"
```

```jsx
  const [password, setPassword] = useState('');
  const [Loading, setLoading] = useState(false);
  const [message, setMessage] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    const data = {
      firstName: firstName,
      lastName: lastName,
      userName: userName,
      emailAddress: emailAddress,
      password: password,
    }
    axios
      .put('/user/editUser/', data)
      .then(res => {
        localStorage.setItem('userInfo', JSON.stringify(res.data));
        setLoading(false);
        setMessage(res.data.message);
        setTimeout(() => {
          navigate('/account')
        }, 2000)
      })
      .catch(err => {
        console.log(err);
      })
  }

  useEffect(() => {
    setFirstName(userInfo.data.firstName);
    setLastName(userInfo.data.lastName);
    setUserName(userInfo.data.userName);
    setEmailAddress(userInfo.data.emailAddress);
    setPassword(userInfo.data.password);
  }, [])

  return (
    <div className="container">
      <div className="row">
        <div className="col-md-6 offset-md-3">
          <div className="card">
            <div className="card-header">
              <h3>Edit Account</h3>
              {message && <Message variant="success">{message}</Message>}
              {error && <Message variant='danger'>{error}</Message>}
              {Loading && <Loading />}
            </div>
            <div className="card-body">
```

```svelte
//AccountEdit states
  let firstName = userInfo.data.firstName;
  let lastName = userInfo.data.lastName;
  let userName = userInfo.data.userName;
  let emailAddress = userInfo.data.emailAddress;
  let password = userInfo.data.password;

  function handleSubmit () {
    const data = {
      firstName: firstName,
      lastName: lastName,
      userName: userName,
      emailAddress: emailAddress,
      password: password,
    }
    axios
      .put('http://localhost:5001/user/editUser/', data)
      .then(res => {
        localStorage.setItem('userInfo', JSON.stringify(res.data));
        setTimeout(() => {
          window.location.href='/Account';
        }, 2000)
      })
      .catch(err => {
        console.log(err);
      })
  }
</script>

<form>
  <div class="form-group">
    <FormGroup>
      <label for="firstName">First Name</label>
      <input
        type="text"
        class="form-control"
        id="firstName"
        placeholder="Enter First Name"
        name="firstName"
        bind:value={firstName}
      />
    </FormGroup>
  </div>
  <div class="form-group">
    <FormGroup>
      <label for="lastName">Last Name</label>
      <input
        type="text"
        class="form-control"
```

```jsx
const handleAddProductSubmit = (e) => {
  e.preventDefault();
  setLoading(true);                                                const category: string
  if (name === '' || description === '' || voucherPrice === '' || imageUrl === '' || category === '') {
    setError('Please fill in all the fields')
    setLoading(false);
  } else {
    axios
      .post('/product/addProduct', { name, description, voucherPrice, imageUrl, category, memberId: userInfo.data._id, claimedSt
      .then((res) => {
        setLoading(false);
        setMessage(res.data.message);
        setTimeout(() => {
          navigate('/Product')
        }, 2000)
      }
      )
      .catch((err) => {
        setLoading(false);
        setError(err.response.data.message);
      }
      );
  }
}
return (
  <><div className="container">
    <div className="row">
      <div className="col-md-6 offset-md-3">
        <h1 className="text-center">Add Product</h1>
        <hr />
        {message && <Message variant='success'>{message}</Message>}
        {error && <Message variant='danger'>{error}</Message>}
        {loading && <Loading />}
        <form onSubmit={handleAddProductSubmit}>
          <div className="form-group">
            <label htmlFor="name">Name</label>
            <input type="text" className="form-control" id="name" placeholder="Enter name" onChange={(e) => setName(e.target.val
```

```svelte
  let name = "";
  let description = "";
  let voucherPrice = "";
  let imageUrl = "";
  let category = "";
  const userInfo = JSON.parse(localStorage.getItem('userInfo'));


  const handleAddProductSubmit = () => {
    if (
      name === "" ||
      description === "" ||
      voucherPrice === "" ||
      imageUrl === "" ||
      category === ""
    ) {
      setError("Please fill in all the fields");
      setLoading(false);
    } else {
      axios
        .post("http://localhost:5001/product/addProduct", {
          name,
          description,
          voucherPrice,
          imageUrl,
          category,
          memberId: userInfo.data._id,
          claimedStatus: false,
        })
        .then((res) => {
          setTimeout(() => {
            window.location.href = "/Product";
          }, 2000);
        })
        .catch((err) => {
          setLoading(false);
          setError(err.response.data.message);
        });
    }
  };
</script>

<div class="container">
  <div class="row">
    <div class="col-md-6 offset-md-3">
      <h1 class="text-center">Add Product</h1>
      <hr />
      <form on:submit|preventDefault={handleAddProductSubmit}>
        <div class="form-group">
```

```jsx
      <div className="row">
        <div className="col-md-6 offset-md-3">
          <h1 className="text-center">Products listed</h1>
          <hr />
          <table className="table table-striped">
            <thead>
              <tr>
                <th scope="col">ProductID</th>
                <th scope="col">Name</th>
                <th scope="col">Voucher Price</th>
                <th scope="col">Product Claimed?</th>
              </tr>
            </thead>
            <tbody>
              {products ? (
                products.map(product => {
                  return (
                    <><tr key={product._id}>
                      <td>{product._id}</td>
                      <td>{product.name}</td>
                      <td>{product.voucherPrice}</td>
                      <td>{product.claimedStatus ? 'Yes' : 'No'}</td>
                      <td><Button variant="primary" onClick={updateClaimedStatus} >Update Claim</Button></td>
                    </tr></>
                  )
                }
                )
              ) : (
                <tr>
                  <td>No Products</td>
                </tr>
              )}
            </tbody>
          </table>
        </div>
      </div>
    </div>
```

```svelte
</script>

<div class="container">
  <div class="row">
    <div class="col-md-6 offset-md-3">
      <h1 class="text-center">Products listed</h1>
      <hr />
      <table class="table table-striped">
        <thead>
          <tr>
            <th scope="col">ProductID</th>
            <th scope="col">Name</th>
            <th scope="col">Voucher Price</th>
            <th scope="col">Product Claimed?</th>
          </tr>
        </thead>
        <tbody>
          {#if products}
            {#each products as product (product._id)}
              <tr>
                <td>{product._id}</td>
                <td>{product.name}</td>
                <td>{product.voucherPrice}</td>
                <td>{product.claimedStatus ? "Yes" : "No"}</td>
                <td>
                  <Button variant="primary" on:click={updateClaimedStatus}
                    >Update Claim</Button
                  >
                </td>
              </tr>
            {/each}
          {:else}
            <tr>
              <td>No Products</td>
            </tr>
          {/if}
        </tbody>
```

```jsx
import axios from 'axios';
import OrderDetailsModal from './OrderDetailsModal';
/* ===================================================================== */
/*                            Layout Section                             */
/* ===================================================================== */
const TokenList = () => {
  const [tokens, setTokens] = useState([]);
  const userInfo = JSON.parse(localStorage.getItem('userInfo'));

  // get user email address from database to update user and send to localStorage
  axios
    .post('/user/', { emailAddress: userInfo.data.emailAddress })
    .then(res => {
      localStorage.setItem('userInfo', JSON.stringify(res.data));
    }
    )
    .catch(err => {
      console.log(err);
    }
    )
 // set token from user information
  useEffect(() => {
    setTokens(userInfo.data.tokens);
  }, [])

  return (
    <div>
      <div className="container">
        <div className="row">
          <div className="col-md-6 offset-md-3">
            <table className="table table-striped">
              <thead>
                <tr>
                  <th scope="col">Tokens</th>
                  <th scope="col">Order Details</th>
                </tr>
              </thead>
              <tbody>
                {tokens ? (
                  tokens.map(token => {
                    return (
                      <tr key={token._id}>
                        <td>{token}</td>
                        <td>
                          <OrderDetailsModal token={token} />      You, last week • got tokens working on beneficiary account …
                        </td>
                      </tr>
                    )
                  }
```

```svelte
<script>
  import axios from "axios";
  import { onMount } from "svelte";
  import OrderDetailsModal from "../components/OrderDetailsModal.svelte";


  const userInfo = JSON.parse(localStorage.getItem("userInfo"));
  let tokens = [];      You, 2 days ago • added beneficiary and members lists of orders and…

  // get user email address from database to update user and send to localStorage
  axios
    .post("http://localhost:5001/user/", { emailAddress: userInfo.data.emailAddress })
    .then((res) => {
      localStorage.setItem("userInfo", JSON.stringify(res.data));
    })
    .catch((err) => {
      console.log(err);
    });

    onMount(async () => {
      tokens = userInfo.data.tokens;
    })
</script>

<div>
  <div class="container">
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <table class="table table-striped">
          <thead>
            <tr>
              <th scope="col">Tokens</th>
              <th scope="col">Order Details</th>
            </tr>
          </thead>
          <tbody>
            {#if tokens}
              {#each tokens as token (token)}
                <tr>
                  <td>{token}</td>
                  <td>
                    <OrderDetailsModal {token} />
                  </td>
                </tr>
              {/each}
            {:else}
              <tr>
                <td>No Tokens</td>
              </tr>
```

**Note: this section of the application will have a redesign as it does not fully meet the requirements. It currently lacks an interface to accept a token. The idea was to have both members and beneficiaries "tick off" and complete the order through their own interfaces. A better streamlined approach should be implemented.*

2.2Framework comparison

The two chosen frameworks covered in this report are MERN and MESN. Both frameworks share similarities in their respective stacks/ frameworks.

Backend

Starting with the backend of both stacks the chosen technology is MongoDB, ExpressJS and NodeJS. These technologies make up the bulk of the stack, the main reason for choosing these technologies is the compatibility of with many different front-end frameworks. This allows for the use of many different code libraries to build a flexible and scalable app for in current scope and future developments.

The use of the same backend for both stacks reduce the time taken for configuration, requests, responses, data flow and connections. This will allow for more time to be spent on the front end.

Environment

Node

This is the backend environment for the application, built on the open-source V8 JavaScript engine. (Codecademy Team, n.d.).

Express

A popular node web framework. It is used for writing handlers for requests, HTTP and URL paths (routes). Setting up common web app settings like port connections, rendering responses and request handling such as “middleware”. (Express/Node Introduction, n.d.).

MongoDB/Mongoose

The database/data storage of the application. Popular, and built with developers in mind it’s easy to get started and install with ease. Mongoose is a JavaScript programming library that creates a connection between MongoDB and Express.

Packages

Alongside the environment a list of packages is installed to meet the requirements of the Stuff and Things Clothing Charity.

NPM

Firstly, node package manager (npm) is the default package manager for the JavaScript runtime environment NodeJS. Npm has the world’s largest online database of packages called the npm registry. (About Npm | Npm Docs, n.d.).

Upon cloning or building a project to a local machine it is recommended to use `npm install` to install dependencies, modules, and the latest version.

Nodemon

A tool that helps development of node-based applications by auto restarting the node application whenever there are file changes to the directory. If any errors occur during development the server will hang but upon fixing it the server will automatically restart.

Dotenv

A module that loads sensitive environment variables from a .env file into a process.env file. It stores this configuration in the environment separate from the code based on twelve-factor app methodology.

Concurrently

This simple package allows multiple commands to run concurrently at the same time. This is useful for running the backend and front end of the web application simultaneously for development.

Bcrypt

A library package to help hash passwords. This is so sensitive information from a user such as a password can be safely stored on the database without fear of that information being leaked to other individuals.

Voucher-code-generator

A library originally from Voucherify, it generates unique, random, hard to guess coupon/voucher/ token codes. It can be used for a multitude of applications but for this web application it will function as tokens for beneficiaries to spend. (Voucher-Code-Generator, n.d.).

Database Connection

For our dataflow throughout the application, there needs to be a place to store data. After creating a database with mongoDB a backend folder is made with a `db.js` file in its root. In the `db.js` the following code is compiled.

```
// use .env to secure sensitive credentials
require("dotenv").config();
//use mongoose to connect between express and mongoDB
const mongoose = require("mongoose");

//function to connect to the database using a try catch.
const connectDB = async () => {
    try {
        // try connection and if successful, console log the connection
        mongoose.connect(process.env.MONGO_URI, {
            useNewUrlParser: true
        });
        console.log("MongoDB connect SUCCESS");
        // if not successful, catch the error and console log the error
    } catch (error) {
        console.log("MongoDB connect FAIL");
        process.exit();
    }
};
//export the function connect
module.exports = connectDB;
```

.env and mongoose are required for connection from express to mongoDB using sensitive passwords and or connection configurations. Whenever `connectDB()` is called the function will run the connect script to determine if the connection to the database was successful or not.

Dotenv

Inside the dotenv file (which is placed in the root of the project folder) sensitive data is contained.

- MONGO_URI – URI provided from MongoDB connection.
- JWT_SECRET – JSON web token for authentication and session use.
- SESSION_SECRET – A key to create the express session.
- SESSION_TIME – Maximum time that a session will last.
- PORT – Port number
  Note: Some of these credentials are not present in the current prototype build but will be in the next iteration of the web application.

Server.js

In the backend folder a `server.js` file should be made this file will import the express module and connection to .env. From here an instance of `express()` can initialised and started with `app.listen`.

```
require("dotenv").config({path: './../.env'});
const express = require('express');
const app = express();
const connectDB = require('./db')
const { notFound, errorHandler } =
require('./middlewares/error.middleware')

// creates express application
app.use(express.json());

// database connect
connectDB();

/* ------------------------------ get routes -----------------------------
*/
const userRoutes = require('./routes/userRoutes')
const productRoutes = require('./routes/productRoutes')
const cartRoutes = require('./routes/cartRoutes')
const orderRoutes = require('./routes/orderRoutes')
const categoryRoutes = require('./routes/categoryRoutes')
```

```
/* --------------------------------- routes ------------------------------
--- */
//list of user
app.use("/user", userRoutes)
//list of product
app.use("/product", productRoutes)
//List of cart
app.use("/cart", cartRoutes)
//List of order
app.use("/order", orderRoutes);
//List of category
app.use("/category", categoryRoutes);

//middleware
app.use(notFound);
app.use(errorHandler);

// request and send to check the backend is running
app.get('/', (req, res) => {
    res.send("API is running");
})
// Use port 5001 and display backend information
app.listen(5001, console.log("Server started on PORT 5001"));
```

Model, View, Controller (MVC)

Three main folders should be created in the backend • Controller • Routes • Models

Model

Models dictate the way in which data is created and stored on the database. Using mongoose, schemas are created/formed to dictate how the model looks and what kind of data will be stored with in the database. In the model objects are created with key and value pairs of information to relate what information and datatypes fit the property of the object in the model. Exporting the schema, we can store this schema as a collection with in the mongoDB database.

```
const mongoose = require('mongoose')
// defines the product schema
const ProductSchema = new mongoose.Schema({
    name: {
        type: 'string',
        required: true,
    },
    description: {
        type: 'string',
        required: true,
    },
    voucherPrice: {
        type: Number,
        required: true,
    },
    countInStock: {
        type: Number,
        required: false,
    },
    imageUrl: {
        type: 'string',
        required: true,
    },
    category: {
        type: Array,
        required: true,
    },
    memberId: {
        type: 'string',
        required: true,
    },
    claimedStatus: {
        type: Boolean,
        required: true,
        default: false
    }
});

const product = mongoose.model("product", ProductSchema);
module.exports = product
```

Controller

The controller receives user input and initiates a response by making calls on the model objects from the schema. Once input is received from the user, it instructs the model and viewport to perform actions based upon the user’s input.

A controller JS file will need to be created and the model imported to relate any user input data. Using async functions along with try, catch, and await to resolve any promises. Mongoose methods can be used to interact with the model e.g., `findOne` and `insertMany` etc. This will create a response with the appropriate status codes, messages and or new data to confirm that the controller is functioning as intended.

```
const Users = require('../model/users');
//@desc   login user
//@route  POST /user/login/:userEmail
//@access Public
//login user
const loginUser = async (req, res, next) => {
    const { emailAddress, password } = req.body;
    try {
        const user = await Users.findOne({ emailAddress: emailAddress });
        if (!user) {
            return res.status(404).json({
                message: 'User not found'
            });
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(400).json({
                message: 'Invalid credentials'
            });
        }
        res.status(200).json({
            success: true,
            data: user,
            message: 'User logged in successfully'
        });
    }
    catch (error) {
        res.status(500).json({
            message: 'Invalid user data'
        });
    }
}
```

Routes

Routes determine the path of the request and how to handle the data. They are endpoints and are usually assigned a task through a function. For example, the task could be logging in the user through a route called `/users/login`.

To consume an API or use an API an HTTP request is made such as `GET` to fetch data. The request is aimed at the server URL/users/login. The response will return an array of objects that can then be used in the frontend.

```
const express = require('express');
router = express.Router();

const {
    registerUser,
    showAccount,
    getUser,
    loginUser,
    logoutUser,
    editUser,
    deleteUser
} = require('../controller/userController')

//@desc   create new user
//@route  POST /user/register
//@access Public
// Register new user
router.post('/register', registerUser)
```

```
//@desc   view account
//@route  GET /user/account/:userEmail
//@access Public
// view account
router.get('/account/', showAccount)

//@desc   get user by email address
//@route  GET /user/:userEmail
//@access Public
// get user by email address
router.post('/', getUser)

//@desc   login user
//@route  POST /user/login/:userEmail
//@access Public
//login user
router.post('/login/', loginUser)
```

In the `server.js` file the routes are declared and used

Frontend

For the frontend to function as intended it is essential that the backend requirements be in place and configured correctly to achieve the desired result.

React

React is a JavaScript library created by Facebook. One of its main uses is for the creation of interactive UIs. Declarative views and encapsulated components make it easy to write predictable code and manage the state of the components easily. (React – a Javascript Library for Building User Interfaces, 2019).

React will be used to handle the view layer of one of the chosen stacks.

Create project

```
npx create-react-app my-app
cd my-app
npm start
```

This creates a barebone intro react webpage. This helps users get situated on working to build their own website.

Component

React defines components as classes or functions. The class component is a traditional way which has built in methods called lifecycle methods. They include `componentDidMount()` and `componentWillUnmount()`.

A function component will be used for this project using react hooks which are new with React 16.8, they let the use of state and other React features without writing a class

```
import React from 'react'
import { Alert } from 'react-bootstrap'

// message component used for displaying error messages. e.g wrong
password.
const Message = ({ variant, children }) => {
  return <Alert variant={variant}>{children}</Alert>
}

Message.defaultProps = {
  variant: 'info',
}

export default Message
```

Another example of hooks.

```
/* -----------------------------------------------------------------------
--- */
/*                              Import Section
*/
/* -----------------------------------------------------------------------
--- */
import React, { useState } from 'react'
import { Link, useNavigate } from 'react-router-dom';
import axios from 'axios'
import Loading from '../components/Loader'
import Message from '../components/Message'
import './Signup.css'
/* -----------------------------------------------------------------------
--- */
/*                              Layout Section
*/
/* -----------------------------------------------------------------------
--- */
const Signup = () => {
  const navigate = useNavigate()
  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');
  const [userName, setUserName] = useState('')
  const [emailAddress, setEmailAddress] = useState('');
  const [password, setPassword] = useState('');
  const [isMember, setIsMember] = useState(false);
  const [isBeneficiary, setIsBeneficiary] = useState(false);
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState('');
  //TODO fix the error handling on selection of both checkbox: Status:
Fixed
  const handleSubmit = (e) => {
    e.preventDefault();
    setLoading(true);
    if (isMember && isBeneficiary) {
      setError('Please select only one of the checkbox')
      setLoading(false);
    }
    if (!isMember && !isBeneficiary) {
      setError('Please select at least one of the checkbox')
      setLoading(false);
    }
    if (isMember && !isBeneficiary) {
      axios
        .post('/user/register', { firstName, lastName, userName,
emailAddress, password, isMember: true, isBeneficiary: false })
        .then((res) => {
          setLoading(false);
          setMessage(res.data.message);
          setTimeout(() => {
            navigate('/login')
          }, 2000)
        })
        .catch((err) => {
          setLoading(false);
          setError(err.response.data.message);
        });
    } else if (!isMember && isBeneficiary) {
      axios
        .post('/user/register', { firstName, lastName, userName,
emailAddress, password, isMember: false, isBeneficiary: true })
        .then((res) => {
          setLoading(false);
          setMessage(res.data.message);
          setTimeout(() => {
            navigate('/login')
          }, 2000)
        })
        .catch((err) => {
          setLoading(false);
          setError(err.response.data.message);
        });
    }
  };

  return (
    <main className="form-signin" onSubmit={handleSubmit}>
      <form>
        <h1 className="h3 mb-3 fw-normal" id="signup-title">Signup</h1>
        {message && <Message variant='success'>{message}</Message>}
        {error && <Message variant='danger'>{error}</Message>}
        {loading && <Loading />}
        <div className="form-floating">
          <input type="first-name" className="form-control"
id="floatingInput" placeholder="FirstName" value={firstName} onChange={(e)
=> setFirstName(e.target.value)} />
          <label htmlFor="floatingInput">First Name</label>
        </div>
        <div className="form-floating">
          <input type="last-name" className="form-control"
id="floatingInput" placeholder="LastName" value={lastName} onChange={(e)
=> setLastName(e.target.value)} />
          <label htmlFor="floatingInput">Last Name</label>
        </div>
        <div className="form-floating">
          <input type="user-name" className="form-control"
id="floatingInput" placeholder="name@example.com" value={userName}
onChange={(e) => setUserName(e.target.value)} />
          <label htmlFor="floatingInput">User Name</label>
        </div>
        <div className="form-floating">
          <input type="email" className="form-control"
id="floatingPassword" placeholder="Password" value={emailAddress}
onChange={(e) => setEmailAddress(e.target.value)} />
          <label htmlFor="floatingPassword">Email Address</label>
        </div>
        <div className="form-floating">
          <input type="password" className="form-control"
id="floatingPassword-signup" placeholder="ConfirmPassword" value=
{password} onChange={(e) => setPassword(e.target.value)} />
          <label htmlFor="floatingPassword">Password</label>
          <div>
            <p>Type of user:</p>
          </div>
        </div>
        <div className="checkbox mb-3">
          <label>
            <input type="checkbox" id="member-check" checked={isMember}
onChange={(e) => setIsMember(e.target.checked)} /> Member
          </label>
          <label>
            <input type="checkbox" id="beneficiary-check" checked=
{isBeneficiary} onChange={(e) => setIsBeneficiary(e.target.checked)} />
Beneficiary
          </label>
        </div>
        <button className="w-100 btn btn-lg btn-primary" id="submit-
button-signup" type="submit">Register</button>
        <Link className="signup-link" to="/Login">Already have an account?
Login</Link>
      </form>
    </main>
  )
}

export default Signup
```

In the example above using `useState()` we have a variable called `firstName` which stores the value of the variable. The function `setName()` is used to update the variable upon an on change event.

Svelte

Svelte is a free open-source software. Much like React, Svelte focuses on building user interfaces. Instead of doing most of the work in the browser like React, Svelte instead builds a compile step that happens upon building the app. Svelte's code surgically updates the DOM when the state of the app changes. (Svelte • Cybernetically Enhanced Web Apps, n.d.).

Svelte will be used to handle the view layer of one of the chosen stacks.

Create project

```
npx degit sveltejs/template my-svelte-project
cd my-svelte-project
npm install
npm run dev
```

Like React this code will install a basic webpage to help users get underway in building their application.

Component

Like React, components are the building blocks of Svelte applications. Instead of .HTML files .svelte is used as a superset of HTML. The following example is a component template, and all three sections are optional.

```
export default Signup

<script>
  // logic goes here
</script>

<!-- markup (zero or more items) goes here -->

<style>
  /* styles go here */
</style>
```

The syntax of a Svelte component makes the time to complete the project shorter than React given the simplicity of it.

```
 <script>
  import { Link, useNavigate } from "svelte-navigator";
  import { Form, FormGroup, FormText, Input, Label } from "sveltestrap";
  import axios from "axios";
  // login states
  let emailAddress = "";
  let password = "";
  let error = "";
  const navigate = useNavigate();

  function handleSubmit() {
    const config = {
      headers: { "Access-Control-Allow-Origin": "*" },
    };
    axios
      .post(
        "http://localhost:5001/user/login/",
        { emailAddress, password },
        config
      )
      .then((res) => {
        console.log(res);
        localStorage.setItem("userInfo", JSON.stringify(res.data));
        setTimeout(() => {
          window.location.href = "/Product";
        }, 2000);
      })
      .catch((err) => {
        error = err.response.data;
      });
  }
</script>

<!------------------------------ Layout Section ----------------------------
-->
<form on:submit|preventDefault={handleSubmit} class="form-signin">
  <h1 class="h3 mb-3 fw-normal" id="login-title">Login</h1>
  <div class="form-floating">
    <FormGroup>
      <input
        type="email"
        class="form-control"
        id="floatingInput"
        placeholder="email address"
        name="emailAddress"
        bind:value={emailAddress}
      />
    </FormGroup>
    <div class="form-floating">
      <FormGroup>
        <input
          type="password"
          class="form-control"
          id="floatingPassword-login"
          placeholder="password"
          name="password"
          bind:value={password}
        />
      </FormGroup>
    </div>
    <button
      class="w-100 btn btn-lg btn-primary"
      id="submit-button-login"
      type="submit">Login</button
    >
    <Link className="signup-link" to="/Signup"
      >Don't have an account? Signup</Link
    >
  </div>
</form>
```

Component creation is easy for beginners to pick up. Svelte has good documentation and is easy to learn due to using HTML and vanilla JavaScript. An important point to consider is that React work can be transferred into Svelte and rebuilt with minimal changes and syntax differences.

Svelte has many options to control and manage state. One of the options used is context API which is perfect for cross-component communication. It allows the passing of data within the component tree using get and set data. Stores are also similar. They differ in the fact that stores are available to any part of the application whereas context is only available to a component and its descendants. It is possible to use the two together as well. (Ugwu, 2020).

3. Recommendation of framework

Both frameworks share similarities in functionality and are both open source. Due to the fact the only key difference here is the frontend of the framework, React or Svelte, a comparison of the two would be best.

React is widely known as the most popular library for building complex and simplistic apps. It has been around since 2011 and has captured the attention of hundreds of businesses, organisations, and developers. Its main purpose is to be fast, scalable, and simple. (Gawkowski, 2022).

React is:

- Search engine optimisation (SEO) friendly
- Stable
- Has a large community and following
- Still growing and evolving
- Provides ready-made components
- Code is scalable and can be reused for future purpose
- Used by many famous companies e.g., Facebook, Amazon, Netflix, and Uber.

Svelte on the other hand is a newer technology only having been around for 7 years when compared to React. It is focused on letting developers build what they need and want. Svelte does not use a virtual DOM in the browser and is often compared to being a compiler. Instead, Svelte complies code into vanilla JS and as a result the code works much faster. (Twardowska, 2022).

Svelte features:

- Simplistic
- Includes styles and animations out of the box
- UI first
- Fast
- No virtual DOM

Svelte has strong benefits but the framework is newer compared to React. This means there is not as large of a following in the way community support and or awareness of the framework. There are a multitude of improvements to be had when it comes to svelte such as more IDE support.

React is well established, stable and has rich community support. With many libraries at its disposal and being compatible with multiple devices it is a safer bet to use for the current purposed web application.

References

About npm | npm docs. (n.d.). Docs.npmjs.com. https://docs.npmjs.com/about-npm

An, E. (2021, December 27). Web frameworks: The complete 2021 beginner's guide. Careerfoundry.com. https://careerfoundry.com/en/blog/web-development/responsiveness-with-a-front-end-framework/

Cairns, C., & Somerfield, D. (2017, January 5). The basics of web application security. Martinfowler.com. https://martinfowler.com/articles/web-security-basics.html

Codecademy Team. (n.d.). What is node? Codecademy. https://www.codecademy.com/article/what-is-node

Express/Node introduction. (n.d.). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Gawkowski, E. (2022, January 28). React vs svelte - which is better for your business in 2022? Pagepro. https://pagepro.co/blog/react-vs-svelte/

Intelegain. (2019, August 6). What are web frameworks and why you need them? Medium. https://intelegain-technologies.medium.com/what-are-web-frameworks-and-why-you-need-them-c4e8806bd0fb

React – A javascript library for building user interfaces. (2019). Reactjs.org. https://reactjs.org

Svelte • cybernetically enhanced web apps. (n.d.). Svelte.dev. Retrieved May 17, 2022, from https://svelte.dev

Twardowska, B. (2022, March 22). Why svelte is the next big thing in javascript development. Naturaily.com. https://naturaily.com/blog/why-svelte-is-next-big-thing-javascript-development

Ugwu, R. (2020, March 20). Application state management with svelte - logrocket blog. LogRocket Blog. https://blog.logrocket.com/application-state-management-with-svelte/

Voucher-code-generator. (n.d.). Npm. Retrieved May 16, 2022, from https://www.npmjs.com/package/voucher-code-generator

What is a web framework? (n.d.). Evolve. https://evolve.ie/q-and-a/what-is-a-web-framework/