# Dan’s Bagel Shop

Group: Foobar

Braysen Goodwin: Back-end Developer
Nathan Johnson: Router
Jared Scott: Front-End Developer

# Functional Requirement 1

## Web-Based and Widely Compatible

-Jared Scott

# Application Front End
# React and JSX

The front end of the application is built using a Javascript library called React. The reasons for this is design choice are as follows:

- Component based design allows for uniformity across different pages.
- The ability for front-end components to be adjusted without requiring a page refresh.
- The state is managed by React freeing the DOM to only handle components.

JSX is a syntax extension of Javascript. The reasons for this design choice are as follows:

- Allows the programmer to write HTML-like elements directly within a React component
- Allows the direct use of CSS styling within React components.
- Shortens time needed to create or edit the appearance of a page.
- JSX gets compiled to regular javascript by Babel, allowing the code to run across many different browsers

# Example of a React Component

Image 1

```
export default function Header({ text }: Props) {
    return (
        <Paragraph>
            <header className="App-header">
                <div className="flex-container" style={{ 'paddingTop': '5px' }}>
                    <div className="flex-child">
                        <p style={{ 'paddingRight': '100px' }}>{text}</p>
                    </div>
                    <div>
                        <img
                            src={logo}
                            className="photo"
                            alt="Dans Bagel Shop"
                            width="275px"
                            height="183px"
                        />
                    </div>
                </div>
            </header>
        </Paragraph>
    )
}
```

Image 2

```
<Header text="Welcome to Dan's Bagel Shop"></Header>
```

Image 3


Image 1 shows how the general component is defined. If a change is made to this definition, the change is immediately made across all web pages using this general component.

Image 2 shows how simple it is to add this component to a new page, the text attribute would be the desired text to display on the final webpage.

Image 3 shows how the element created in image 2 is rendered to the user.

# Functional Requirement 2

## Initializing and Tracking inventory

-Nathan Johnson

# MongoDB, Axios, and JSON

MongoDB is a fast, lightweight, and freely available database system which can be used together with express to create a scalable server.

To manage the shop inventory, the server stores the information within a Mongo database. The manager of the shop can enter inventory items one at a time, or a default inventory can be used which is based on the requirements specified in the requirements definition. *Axios,* an HTTP client,  is used to handle post requests containing updated information. The front-end sends to, and receives information from the back-end in the form of JSON.

# Default inventory and update function.

Image 4

```
var default_inventory = [
    { name: 'Plain', category: 'BAGEL', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
    { name: 'Onion', category: 'BAGEL', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
    { name: 'Cinnamon raisin', category: 'BAGEL', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
    { name: 'Sesame', category: 'BAGEL', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
    { name: 'Cheesy', category: 'BAGEL', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
    { name: 'Pumpernickel', category: 'BAGEL', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
    { name: 'Plain', category: 'SMEAR', quantity: 100, price: 100, onMenu: true, targetCount: 50 },
    { name: 'Honey Nut', category: 'SMEAR', quantity: 100, price: 100, onMenu: true, targetCount: 50 },
    { name: 'Strawberry', category: 'SMEAR', quantity: 100, price: 100, onMenu: true, targetCount: 50 },
    { name: 'French Onion', category: 'SMEAR', quantity: 100, price: 100, onMenu: true, targetCount: 50 },
    { name: 'Bacon', category: 'SAMMICHE_TOPPINGS', quantity: 100, price: 100, onMenu: true, targetCount: 50 },
    { name: 'Egg', category: 'SAMMICHE_TOPPINGS', quantity: 100, price: 200, onMenu: true, targetCount: 50 },
```

Image 5

```
function updateItem(item: any, queryCache: any) {
    axios
        .post(`http://localhost:8100/inventory/${item._id}`, item)
        .then(() => {
            console.log('successfully updated item')
            queryCache.invalidateQueries('inventory')
        })
        .catch((err) => {
            console.log('failed to update item')
            console.error(err)
        })
}
```

Image 4 shows a portion of the default inventory based on requirement 2 within the requirements definition document.

Image 5 is the function used when updating an item within the inventory. Within this function the information regarding the item (quantity, price, etc..) is contained within a JSON called ‘item’. This is then sent as a post request to the server using Axios.

# Functional Requirement 3

## User Authentication and Account Management

-Braysen Goodwin

# User Authentication and Account Management

Account management is a crucial aspect of any web based application. As detailed in the requirements definition document,  users are able to create accounts and alter account information (name, password, add funds, etc…).

Depending on the roles which an account fills, different information is displayed to the user. Image 6 below shows an example of how different navigation bar elements are displayed depending on the roles that an account is assigned.

Image 6

```
{ roles.has("MANAGER") || roles.has("ADMIN") || roles.has("CHEF") ? <NavElement onClick={() => history.replace('/inventory')}>Inventory</NavElement> : null}
{ roles.has("MANAGER") || roles.has("ADMIN")  ? <NavElement onClick={() => history.replace('/analytics')}>Analytics</NavElement> : null}
{ roles.has("MANAGER") || roles.has("ADMIN")  ? <NavElement onClick={() => history.replace('/users')}>Users</NavElement> : null}
{ roles.has("CASHIER") ? <NavElement onClick={() => history.replace('/cashier')}>Cashier</NavElement> : null}
{ roles.has("CHEF") ? <NavElement onClick={() => history.replace('/chef')}>Chef</NavElement>: null}
```

# User Authentication Function

Image 7

```
import type { Roles, MaybeUserRequest } from '../utils/types'

/**
 *  verifiedUserSignedIn - express middleware - verify that the request is coming from a user which is signed in
 *
 * @param  {MaybeUserRequest<>} req - express request
 * @param  {express$Response} res - express response
 * @param  {express$NextFunction} next - express next function
 */
export function verifiedUserSignedIn(
    req: MaybeUserRequest<>,
    res: express$Response,
    next: express$NextFunction
) {
    if (req.user) {
        return next()
    }
    res.status(401).end()
}
```

Image 7 shows the function used to verify that a user is a verified user.

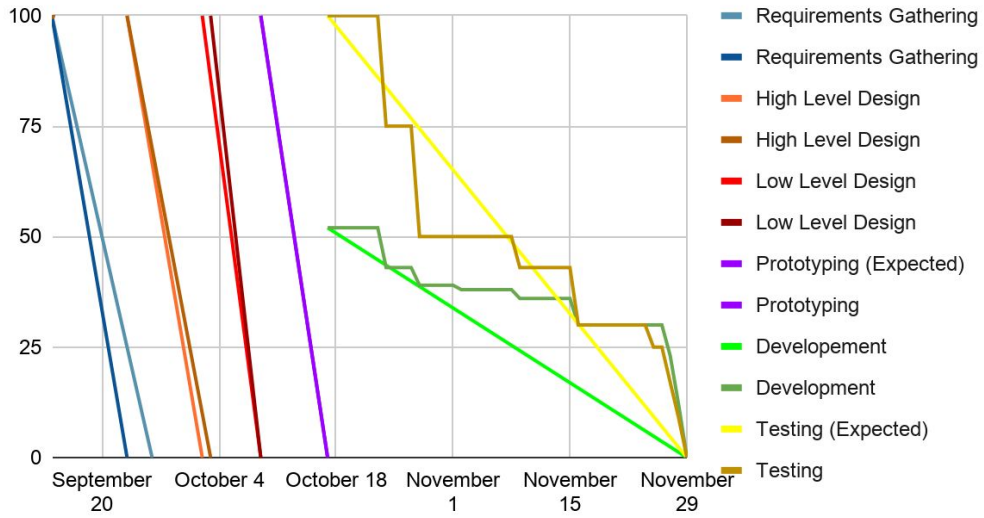
# Use Case Video #1

# Use Case Video #2

# Use Case Video #3

# Use Case Video #4

## Burn Down




Sprint 1: 100% tasks done/expected done
Sprint 2: 40% tasks done/expected done
Sprint 3:  30% tasks done/expected done
Sprint 4: 35% tasks done/expected done
Sprint 5: 100% tasks done/expected done

# References:


4.x API. (n.d.). Retrieved November 29, 2020, from https://expressjs.com/en/api.html

Axios. (n.d.). Axios/axios. Retrieved November 29, 2020, from https://github.com/axios/axios

Documentation. (n.d.). Retrieved November 29, 2020, from https://flow.org/en/docs/

Get started with MongoDB. (n.d.). Retrieved November 29, 2020, from https://docs.mongodb.com/

Thinking in React. (n.d.). Retrieved November 29, 2020, from https://reactjs.org/docs/thinking-in-react.html

Yudi43. (n.d.). Yudi43/React---The-Complete-Guide--incl-Hooks--React-Router--Redux-. Retrieved November 29, 2020, from https://github.com/yudi43/React---The-Complete-Guide--incl-Hooks--React-Router--Redux-

Questions?