INFO30005 Deliverable 3 – Backend with Frontend (Due @11:59pm May 7, 2021)

Before commencing work for this deliverable, you should have covered the relevant lectures and workshops in weeks 1-8. Each group is required to implement the complete front and back end for the functionalities described below.

Features to deliver

Your group needs to implement the frontend and backend to support the following **customer** features (you are not required to implement any vendor features for this deliverable):

1) **Customer Login**
   Your Customer App should have a login form. As per the project requirements document, to place an order, customers need to be logged in. To test the login feature, you can create a few dummy customers in the Mongo database or optionally create a separate form so that a new customer could register. In any case, include the login details of one dummy customer in the README for the markers to test the login feature (create a section called LOGIN DETAILS in the README, and include the login details there).

2) **View menu of snacks (including pictures and prices)**
   Your Customer App should have a user interface that displays the list of snacks. Given that we have a small number of snacks, you may want to show all snacks on the same page (however, this is a suggestion only; you can get creative with how you display the menu to the customers).

3) **Order three different snacks**
   There are different ways of letting a customer choose the snacks the customer wants. You can choose one snack at a time, building up a shopping cart that is stored on the server, or the customer could select multiple snacks and quantities on the same webpage. We will not dictate one way or the other. You can get creative with how your group wants to handle how the customer orders their snacks.

   Irrespective of how the customer makes the snack selections, we need groups to follow the same process to confirm and submit the order to the vendor. After the customer finishes choosing their three snacks, the app should ask the customer to *confirm* the order. If the customer confirms the order, the order must be persisted in the database. Your customer app will be tested on a phone-sized screen and a laptop-sized screen.

4) **View Order Details**
   After placing an order, the customer can view the details of orders that were placed. At this stage, list all orders, fulfilled and outstanding. Since we do not have a vendor app, we are assuming that all orders will be outstanding.

As stated previously, for simplicity, we expect that your group will implement both customer and vendor features in the same backend server and in the same frontend. Use different routes to separate these. For example, customer routes could be under [server-name]/customer and vendor routes under [server-name]/vendor. The suggested technology to implement the frontend is Handlebars. However, groups may choose a frontend JavaScript framework or library such as React.js. (If a group wishes to use a non-taught technology, all members of the group must agree, and you should discuss this decision with your tutor.) We will cover the basics of Handlebars and React in lectures.

In terms of the repository, you can use the repository that you used for Deliverable 2 for the backend. For groups using React, we will give you the option of using a separate repository for the frontend. In case your group decides to use two repositories, we ask your group to use the repository from Deliverable 2 for the backend, and a new one will be provided by your tutors for the frontend (use the following GitHub Classroom link to create the repository for the frontend IF you are using two repositories: https://classroom.github.com/g/7MQ0iQOP].

Database

To support the above features, your group is required to use MongoDB running on Atlas. When marking the above features, we will also check whether the database is updated appropriately.

Note that your database will need to contain sample data that is not directly manipulated by the routes listed above. For example, for customers to order snacks, you (the developer) must have already entered the list of available snacks into the database. (This list is in the business requirements.) For such data, your group can enter values directly into the database through a MongoDB administrator interface such as Compass.

Submission

Closer to the submission date, we will open the submission link. One of your group members will submit via the LMS (i.e. we expect one submission per group):

1. The URL for your live website. The URL should allow us to access your frontend. We will access the backend using the GitHub classroom.
2. Include the name of each GitHub repository that has been used to implement the frontend and backend.
3. For each repository, the commit id that you want us to mark (note that the first 7 characters are sufficient, but you can also provide the full commit id). You may

continue working on your project after submitting the deliverable. The commit id will tell us which commit we should use for marking.

4. Provide the access details to your database (in the README file).

Please keep track of the contribution of each group member to the submission. At the end of semester, each student will be offered the opportunity to comment on the contributions of their group members.

If you are using a frontend framework such as React or Vue, you have two options for managing the repository and hosting of the frontend and backend.

1. The first option is to use only one GitHub repository and host the frontend and backend on the same server. To learn how to do this, view one of the two online tutorials:

https://daveceddia.com/deploy-react-express-app-heroku/ (Links to an external site.) OR

https://www.freecodecamp.org/news/how-to-deploy-a-react-app-with-an-express-server-on-heroku-32244fe5a250/ (Links to an external site.)

If you plan to use the first option, then there are no changes in terms of D3 submission requirements. Simply host the front and back end on the same server and provide the app's URL and the link to the GitHub repository.

2. The second option is that we can give your group a 2nd GitHub repository.

Rubric

Criteria	Excellent	Good	Inadequate	Not Submitted (Marks: 0.0)
Core features – Backend	The backend core functionalities identified in D3 are fully built using Express. The website content is loaded from an appropriate database. **30**	The core usage scenarios are mostly implemented, but there are still large gaps. **20**	The content of the core usage scenarios partially loads from an appropriate database. **10**	The core usage content is static. **0**
Core features - Frontend	The frontend of the core functionalities identified in D3 are fully built. Makes use of Handlebars, React or another appropriate frontend framework or template engine. **15**	The core usage scenarios are almost implemented. **10**	The core usage scenario is partially implemented. **5**	The core usage scenario is not implemented. **0**
Presentation (Frontend)	Professional look and feel, excellent use of colour, type, and layout, responsive design, appropriate image size/resolution for different platforms. **10**	The interface looks fine, but could still be substantially improved. **5**	Inadequate use of colour, type, and layout **3**	The website does not load **0**

Usability (Frontend)	Seamless usability in the core usage scenario. App is easy to use with very few unexpected outcomes. Errors and data validation are handled well. Good Information architecture. **15**	The system is mostly usable, but still present usability problems. When interacting with the App, some aspects are confusing and/or need more effort than warranted **10**	The system has severe usability problems. Very little to any of the App is intuitive to use. **5**	The App does not load **0**
Separation of Concerns	Presentation (View), behavior (routes and controllers) and content (database) are properly separated **10**	Concerns are mostly separated, but there are issues **5**	Some concerns are separated, but there is substantial overlap **3**	Presentation, behaviour and content are not separated, **0**
Repository - Code	A high standard of coding and commenting is evident, the code is easy to read and appears clear to maintain, ready to be open sourced. Git history. **20**	Reasonable clarity and code can be followed with some effort, comments help to understand the code, logical organization may be lacking in some ways and there may be a number of questionable expressions **10**	Extremely hard to read, little or no attention to code formatting, lack of logical organization, little or no appreciation of coding and commenting practices **5**	The code is not in the repository **0**