**Narrative for Enhancement Two**

**Briefly describe the artifact. What is it? When was it created?**

The artifact used in enhancement two is the Travlr GetAways web application, which was developed in February of 2024. The web application was developed in a previous project and while the web application met all of the requirements for the project, it was still far away from meeting the client requirements. Once the web page is fully completed, admins should be able to use CRUD methods to add, edit, and delete trips, rooms, meals, and more from the admin facing page and those changes should be reflected on the customer facing page. Additionally, login, registration, and checkout features should be implemented on the customer facing web page as well. Prior to any enhancements the web application has an admin page and a customer facing web page with the only edit-able item being trips. Admins can login and add, edit, or delete a trip and those changes are reflected in the database and on the customer facing web application. In enhancement one a rooms link was added to the admin page as well as an add room, edit room, and delete room button fulfilling the software enhancement requirement, but in order for the buttons to be fully functional enhancement two and enhancement three must be completed.

**Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?**

As previously mentioned, the artifact used for this enhancement will be the Travlr GetAways web application. This artifact was recently developed but still needs a lot of work to bring it up to standards with the client's wireframe. In this enhancement, the data structure will be enhanced by restructuring the customer facing rooms page to load the database dynamically instead of pulling information from the public static .html file. In doing so, the customer facing page will display changes made to rooms in the admin page. The rooms portion will be restructured in this enhancement by refactoring the code to use a MVC(Model, View, Controller) structure. Additionally, an API endpoint for localhost:3000/api/rooms will be added for testing purposes and easier compatibility with POSTMAN. Implementing the MVC structure for 'rooms' brings the project one step closer to completion as it integrates the database with both the admin page and the customer facing page. As is, the data for rooms was hard coded into the HTML file and the data was displayed in that way. Unfortunately, that setup is not practical for Travlr GetAways because the html code would need to be changed any time the admin team decided they wanted to add, edit, or delete a trip. Setting the rooms portion up to load the database dynamically to display real live data instead of hard-coded html data is crucial to meeting the client's requirements for the final project.

Restructuring the data for this enhancement showcases the following skills and abilities :

***Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices (data structures and algorithms)***

This course outcome was met by improving the data structure by refactoring rooms to use an MVC structure instead of pulling static .html information.

***Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals (software engineering/design/database)***

This course outcome was met by integrating the database into the MVC structure so that changes made on the admin SPA page will be reflected immediately on the customer facing page once CRUD methods are completed (in the next enhancement).

***Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources***

This course outcome was met by ensuring that crucial elements of the database, such as add edit and delete functions are only available to users that are logged in with authorized credentials.

**Did you meet the course objectives you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?**

Planned course objectives were met in this enhancement as described above. Updates to the outcome-coverage plan involve adding CRUD functionality to the add room, edit room, and delete room buttons that are currently displayed on the Angular SPA admin page when an authorized user logs in. After the CRUD functionality methods are implemented in the next enhancement, authorized admin users will be able to manipulate both trips and rooms and the changes made to the database will be displayed immediately on the customer facing Express page. At the end of all three enhancements the rooms portion of the web application will be fully functional on the admin page and reflected on the customer face. These enhancements bring the project one step closer to meeting client requirements for a completed project.

**Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

Enhancing and modifying this artifact just prioritized again how important it is to have neat, easy to read, and easy to duplicate code. Since my original code for trips was so neat it was easy to make the necessary changes to re-create the MVC structure for rooms as well. Having easy to re-use code made this enhancement so much easier to implement.

Fortunately, I did not have many challenges with this enhancement. I initially forgot to add the rooms router in the app.js file, but once I stepped away from my code and returned with a fresh mind, I was able to quickly correct the issue. See screen grabs below of how the rooms portion of the Express customer facing page was loading prior to this enhancement, and how it's loading now.

Prior to this enhancement, rooms loaded by pulling the static .html file in the public folder of our web application. Any changes made to rooms in the database would not appear on the customer facing page unless the same data was changed in the .html file. The page was rendered like this :

A screenshot of a computer

Description automatically generated

If a user attempted to navigate to localhost:3000/rooms the following page rendered :

A screenshot of a computer

Description automatically generated

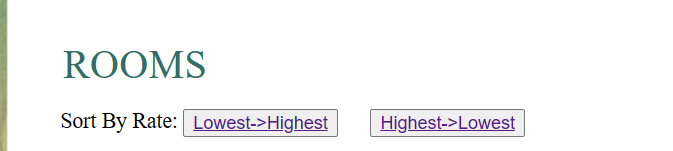
Now that the rooms data has been restructured to load dynamically using the MVC structure the following page renders :

A screenshot of a hotel room

Description automatically generated

The data is now pulled from the MongoDB instead of the public .html file and data changes made on the admin site will be reflected on the customer site as well.

Additionally, I implemented a price sorting feature for the rooms displayed on the customer facing site. To do this, I first created API endpoints for two new pages , “/SortUp” and “/SortDown”. I then used MongoDB’s built in sort() algorithm to sort the rooms by rate in either ascending or descending order based on the way the user chooses to sort the rooms. If a user clicks sort price from lowest by highest, the web application will render rooms that were sorted using sort(asc) and will be rendered on a new web page that I have titled “SortUp”. The interface remains the same for a seamless user experience. If a user clicks sort price from highest to lowest, the web application will render rooms that were sorted using sort(desc) and will be rendered on a new web page that I have titled “SortDown”. Although a new web page is technically rendered, the user will not notice any difference on the web page itself other than the order that the rooms are sorted. The sort filter options looks like this :



If a user clicks “Lowest->Highest” the rooms will be rendered from lowest rate to highest rate. As shown below. Notice the URL has now changed to our new URL endpoint /SortUp A screenshot of a web page

Description automatically generated

To render the SortUp page, I implemented the sort(asc) algorithm in the controller file that’s located in the App\_Api folder (**travlr\app\_api\controllers\sortUp.js**). See the function below :

A screenshot of a computer program

Description automatically generated

In a similar manner, the SortDown endpoint is rendered when a user chooses to sort from “Highest->Lowest”. When a user clicks “Highest->Lowest”, our new endpoint /SortDown is rendered and rooms are sorted from highest rate to lowest rate as seen below:

A screenshot of a room

Description automatically generated

To sort the rooms from highest rate to lowest rate, I implemented MongoDB’s built in sort() algorithm in the SortDown controller that’s located in our App\_Api folder (**travlr\app\_api\controllers\sortDown.js**). See the function shown below :

A computer code with many colored text

Description automatically generated with medium confidence

For this enhancement the following folders and files were added or modified :

The following files were added in the app\_api folder :

**\travlr\app\_api\routes\rooms.js**

**\travlr\app\_api\controllers\rooms.js**

**\travlr\_final\travlr\app\_api\models\room.js**

**\travlr\app\_api\controllers\sortUp.js**

**\travlr\app\_api\controllers\sortDown.js**

**\travlr\app\_api\routes\sortUp.js**

**\travlr\app\_api\routes\sortDown.js**

The following files were added or modified in the app\_server folder:

**\travlr\app\_server\controllers\sortUp.js**

**\travlr\app\_server\controllers\sortDown.js**

**\travlr\app\_server\routes\sortUp.js**

**\travlr\app\_server\routes\sortDown.js**

**\travlr\app\_server\views\rooms.hbs** – Edited handlebars file to include the sort price option

Additionally lines 33-37, lines 58-62, and lines 73-78 were added in the **\travlr\app.js** file and lines 19-21 , and lines 49-88 were added in the **\travlr\app\_api\routes\index.js** file.