The Marence Bistro
Project Group #69
Team Members: Mazen Abdullah, Terence Tang
Deliverable: Project Step 7 (Portfolio Assignment)
Date: Dec 7, 2020

Website URL: http://flip2.engr.oregonstate.edu:3509/

Executive Summary:

Over the past few months, we've really been able to polish our portfolio project site due to the peer review feedback from piazza. The use of peer review enabled us to get a diverse set of feedback ranging from structural design, implementation tips, UI and ease of use comments, etc. which gave us a wide view of what could be done to improve our application.

During the initial stages of the project, one of the more formative updates was the introduction of intersection tables to map out M:M relationships and enable us to better manage those relationships down the road.

Another piece of feedback highlighted the types of relationships we were defining as part of our DDL queries, which restricted nullable relationships. We revisited which relationships we thought would require partial vs. full participation and updated our DDL queries as needed. These small participation tweaks continued as we built out our project website We realized some relationships were causing unnecessary constraints on our application's functionality e.g. forcing full participation of Chefs to our Chef Schedule (a chef does not always need to be scheduled for work).

Having a wide range of folks looking at our code was also great for catching details like incorrect data type's defined vs. in our ERD and documentation. Peer feedback also helped us catch the need for defining Unique and Primary keys in our tables, specifically in the tables where users would be adding new records constantly and in our intersection tables. We enabled unique constraints in our tables that allowed users to add ingredients and menu items based on ingredient and menu item names and concatenated primary keys for our intersection tables. This allowed us to structure our database to conform to 3rd Normal Form.

We also spent some time working on our front-end UI and user experience given Week 5 / 6 feedback. We implemented drop-down options for foreign key selections and provided named options vs. requiring users to enter 'ID' numbers for records they wanted to add/edit. We also added hidden forms on our main HTML pages so that adding and editing entries could happen on the same page without having to load up additional pages for different functions.

Finally, we also spent quite some time during the past few weeks implementing methods of data validation and error handling for our site to further enhance the user experience, simplifying our defined process of ‘restaurant management’ for our application. This includes..

- Dynamically produced drop-down selection lists based on other input values e.g. if two drop downs are provided, selection of the first drop-down affects options in the second drop-down (for management of M:M intersection table/view)
- Cascading updates/deletes to additional tables beyond the scope that is handled by mySql during DDL formation of the DB. e.g. if a ‘Cuisine’ is deleted, how does this affect potentially unrelated tables like Chef Schedules?
- Implementing a way to ‘reset’ the sample database provided for testing and grading purposes with a ‘Reset DB’ button on the index (‘/’) page.

Project Outline

The Marence Bistro is a restaurant that serves lunch and dinner 7 days a week. For each day of the week Marence serves a specific cuisine, with a total of 3 different cuisines (1 cuisine for 3 days a week, 2 cuisines for 2 days a week). Each chef at the restaurant specializes in only one type of cuisine. The database required will keep track of the inventory of ingredients, which ingredients are used in which dishes served at the restaurant, which chefs are capable of cooking them, and what the chef schedule is for the restaurant to ensure that the restaurant is operational and has the right level of coverage needed.

Database Outline

Entities:

- Ingredients
- MenuItemIngredients
- MenuItems
- Cuisines
- Chefs
- RestaurantSchedule
- ChefSchedule

Entity: Ingredients

Details: Records the raw ingredients used by the restaurant
Attributes:

- ingredientID = int, auto_increment, unique, not NULL, PK
- ingredientName = varchar(255), not NULL
- isVegan = bool, not NULL
- Inventory = int

Relationship:

- 1:M relationship between Ingredients and MenuItemIngredients is implemented with ingredientID as FK inside MenuItemIngredients
- M:M relationship between Ingredients and MenuItems is implemented with MenuItemIngredients as an intersection table / composite entity

Entity: MenuItemIngredients
Details: Records the ingredients used for each menu item for the restaurant
Attributes:

- menuItemID = int, not NULL, FK
- ingredientID = int, not NULL, FK
- Composite Primary Key = (menuItemID, ingredientID) unique, not NULL, PK

Relationship:

- M:1 relationship between MenuItemIngredients and Ingredients is implemented with ingredientID as FK inside MenuItemIngredients

- M:1 relationship between MenuItemIngredients and MenuItems is implemented with menuItemID as FK inside MenuItemIngredients

Entity: MenuItems
Details: Records the menu items for the restaurant and the category of cuisine it relates to
Attributes:
- menuItemID = int, auto_increment, unique, not NULL, PK
- menuItemName = varchar(255), not NULL
- cuisineID = int, not NULL, FK
- price = decimal(10,2) , not NULL

Relationship:
- 1:M relationship between MenuItems and MenuItemIngredients is implemented with menuItemID as FK inside MenuItemIngredients
- M:M relationship between Ingredients and MenuItems is implemented with MenuItemIngredients as an intersection table / composite entity
- M:1 relationship between MenuItems and Cuisines is implemented with cuisineID as FK inside MenuItems

Entity: Cuisines
Details: Records the type of cuisine that is served at the restaurant.
Attributes:
- cuisineID = int, auto_increment, unique, not NULL, PK
- cuisineName = varchar(255), not NULL

Relationship:
- 1:M relationship between Cuisines and MenuItems is implemented with cuisineID as a FK inside of MenuItems
- 1:M relationship between Cuisines and RestaurantSchedule is implemented with cuisineID as a FK inside of RestaurantSchedule
- 1:M relationship between Cuisine and Chefs is implemented with cuisineID as a FK inside of Chefs.

Entity: Chefs
Details: Records the chefs and the types of cuisine they can cook for the restaurant
Attributes:
- chefID = int, auto_increment, unique, not NULL, PK
- firstName = varchar(255), not NULL
- lastName = varchar(255), not NULL
- cuisineID = int, not NULL, FK

Relationship:
- M:1 relationship between Chefs and Cuisine is implemented with cuisineID as a FK inside of Chefs
- 1:M relationship between Chefs and ChefSchedule is implemented with chefID as a FK inside of ChefSchedule

- M:M relationship between Chefs and RestaurantSchedule is implemented with ChefSchedule as an intersection table / composite entity

Entity: ChefSchedule
Details**:** Records a weekly schedule of on-duty chefs by day of the week.
Attributes
- dayofWeek = varchar(255), not NULL, FK
- chefID = int, FK

Relationship:
- M:1 relationship between ChefSchedule and Chefs is implemented with chefID as a FK inside of ChefSchedule
- M:1 relationship between ChefSchedule and RestaurantSchedule is implemented with dayofWeek as a FK inside of ChefSchedule

Entity: RestaurantSchedule
Details: Records a weekly schedule of on-duty chefs by day of the week.
Attributes
- dayofWeek = varchar(255), not NULL, FK
- cuisineID = int, FK

Relationship:
- M:1 relationship between RestaurantSchedule and Cuisines is implemented with cuisineID as a FK inside of RestaurantSchedule
- 1:M relationship between RestaurantSchedule and ChefSchedule is implemented with dayofWeek as a FK inside of ChefSchedule
- M:M relationship between Chefs and RestaurantSchedule is implemented with ChefSchedule as an intersection table / composite entity

HTML Interface

Home / Index - Readout of this week's schedule with each day listing the cuisine and chefs assigned.
- Search: Searches a chef's shifts using user entries for First and Last Names.

Ingredients Page - Add / Delete ingredients from list or update inventory levels
Database:
- Add: Adds new record of a new ingredient to Ingredients table.
- Delete: Deletes an existing record of an ingredient in Ingredients table.
- Update: Updates an existing record of an ingredient in Ingredients table.

Cascaded Affects:
- On Add: N/A
- On Delete: Removal of all related menuItems and menuItemsIngredients records
- On Update: Cascades updates to the menuItemIngredients table

MenuItems Page - Add / Delete / Update Menu Items. Adding or updating will allow edits to ingredients used and what type of cuisine it aligns with.
Database:
- Add: Creates a new record of a menu item to MenuItems table. Asks for related ingredients and cuisine
- Delete: Deletes an existing record of a menu item from MenuItems table.
- Update: Updates an existing record of a menu item in the MenuItems table. Asks for updated related ingredients.

Cascaded Effects:
- On Add: Creates associated records in the MenuItemIngredients intersection table for related ingredientIDs and menuItemIDs
- On Delete: Deletes associated records in the MenuItemsIngredients intersection table based on FK menuItemID
- On Update: Deletes or creates associated records in the MenuItemIngredients intersection table for related ingredientIDs and menuItemIDs

Cuisines Page - Add / Delete / Update cuisine items.
Database:
- Add: Adds new record of a new cuisine to the Cuisines table
- Delete: Deletes an existing record of a cuisine from the Cuisines table
- Update: Updates an existing record of a cuisine from the Cuisines table

Cascaded Effects:
- On Add: N/A
- On Delete:
    - Deletes associated records in the MenuItems table based on FK cuisineID
    - Nulls cuisineID data in associated records in the RestaurantSchedule table based on FK cuisineID
    - Nulls cuisineID data in associated records in the Chefs table based on FK cuisineID
- On Update: N/A

Restaurant Schedule Page - Aligns Cuisine to Day it will be featured.
Database:

- Add: Adds a new record of a new restaurant day schedule to the RestaurantSchedule table. Asks for what cuisine is related to what day.
- Delete: Deletes an existing record of a restaurant day schedule from the Restaurant Schedule table.
- Update: Updates an existing record of a restaurant day schedule from the Restaurant Schedule table.

Cascaded Effects:

- On Add: N/A
- On Delete: Deletes associated records in the ChefSchedule table based on FK dayOfWeek
- On Update: Cascades dayOfWeek update to ChefScheduleTable

Chefs Page - Add / Delete / Update Chefs. Adding and Updating allow for edits to cuisine alignment with chefs.
Database:

- Add: Adds a new record of a new chef to the Chefs table. Asks for associated cuisineID.
- Delete: Deletes an existing record of a chef from the Chefs table
- Update: Updates an existing record of a chef from the Chefs table.

Cascaded Effects:

- On Add: N/A
- On Delete: Nulls chefID data in associated records in the ChefSchedule table based on FK chefID
- On Update: N/A

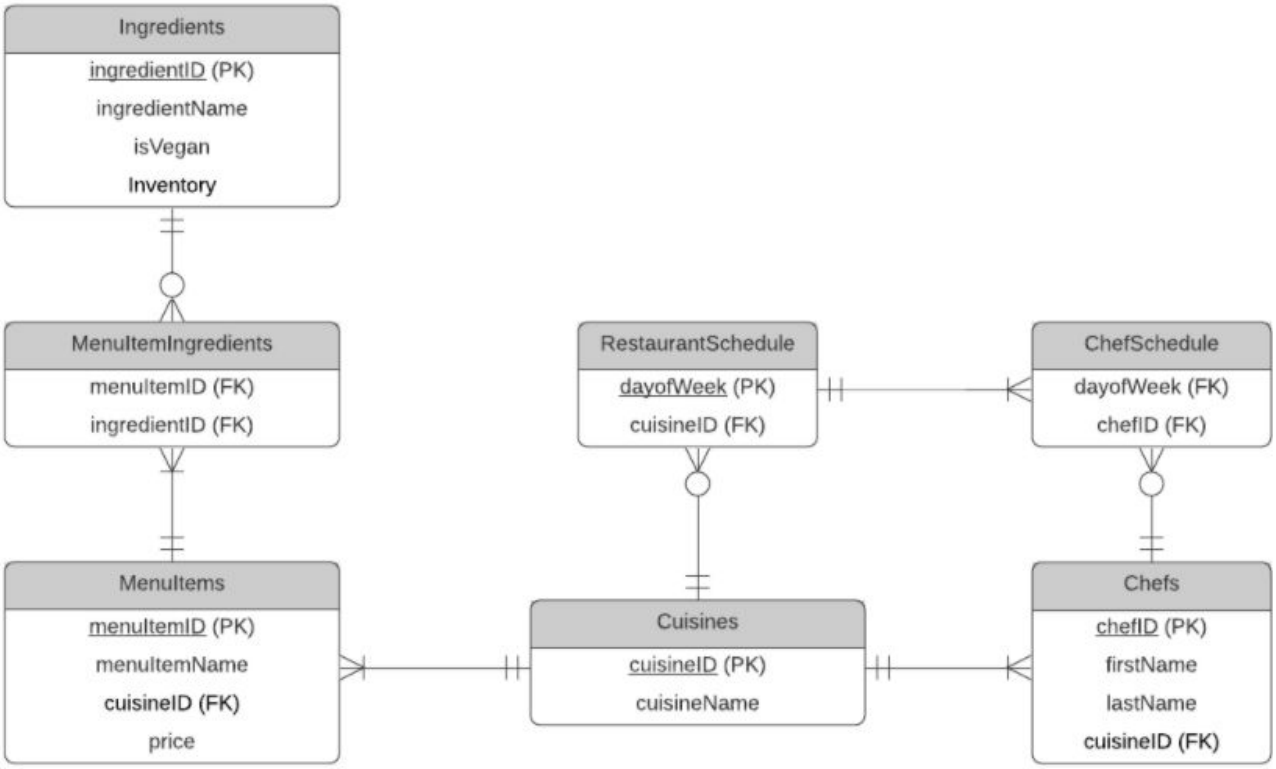
Chef Schedule Page - Aligns Chef to Day they are scheduled to work on.
Database:

- Add: Adds a new record of a newly scheduled chef in the ChefSchedule table
- Delete: Deletes an existing record of a scheduled chef in the ChefSchedule table
- Update: Updates an existing record of a scheduled chef in the ChefSchedule table

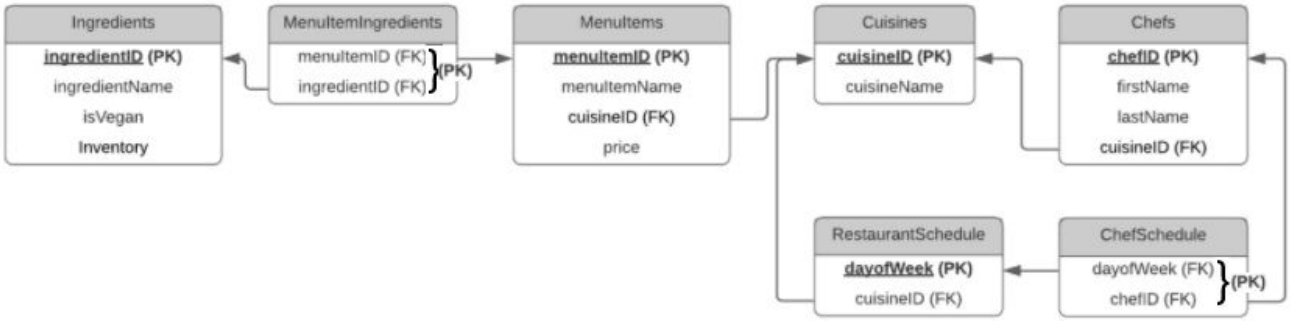
Cascaded Effects:

- On Add: N/A
- On Delete: N/A
- On Update: N/A

Entity Relationship Diagram (ERD)



Schema



Database Normalization

After completing the database schema, data normalization was made to check the normal form of the database. Upon examination, it was shown that the current database is in 3rd Normal Form (3NF). The requirements of 3NF (of which are satisfied by the database) are shown below.

- Every value is a non-divisible atomic value
- Transitive dependencies are valid
- No nonprime attribute is transitively or partially dependent on any key.
- Attributes are functionally dependent on entire keys.

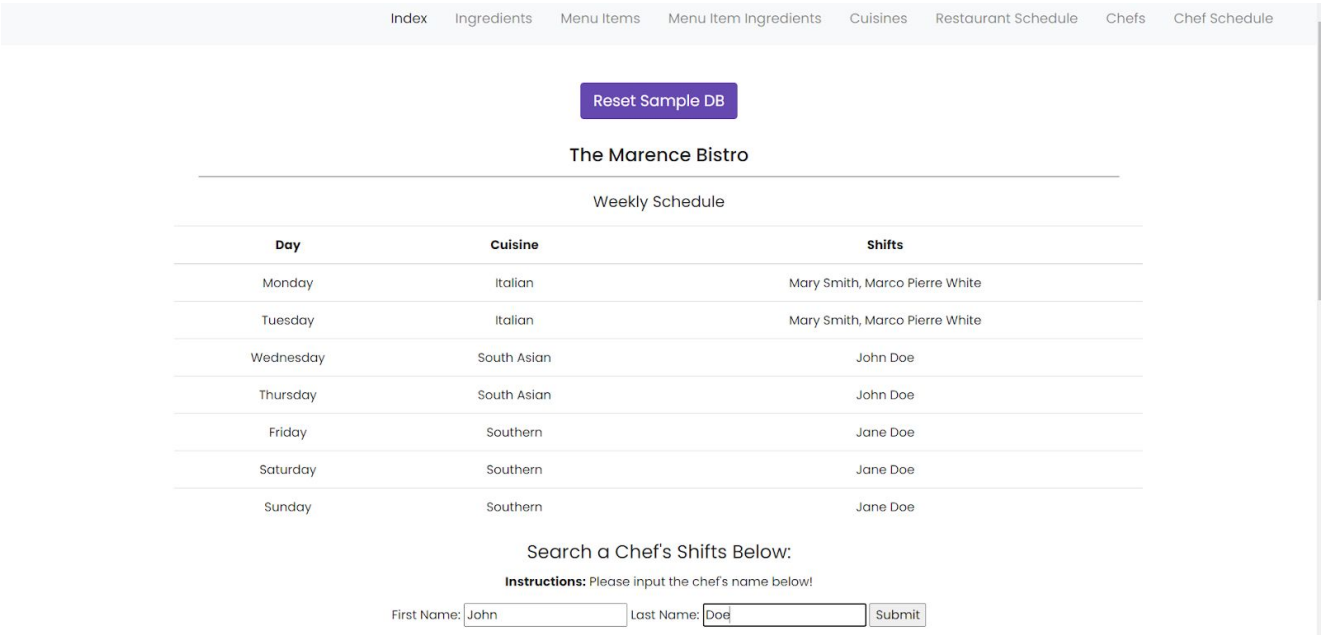
Updates Based on Feedback (TRACKING):

Step	Feedback (Verbatim)	Updates Implemented
Project Step 6	When I went to add something to the "menu item ingredients" page, it asked me for ID numbers. So it might be a good idea to reconfigure this if you have time, so that a user wouldn't have to go back and forth to copy down the ID numbers. Perhaps it should display all the possible options in a drop down of some kind.	Added drop-downs for foreign key selection instead of using 'ID' numbers. Enabled all adding and editing forms to happen on the same http page; reduces number of pages served and better UI experience.
Project Step 5	Based on the code you already have written, I would continue what you were already doing. The read queries were easy to understand and seem to work beautifully, as well as the html pages look good. I'm not familiar with how to do things in flask. It really helped me to test all of the backend functionality first and write small tests to make sure CRUD worked for each table. Once the backend was working it was easier for me to build a frontend and link the two as opposed to trying to build and test the frontend and the backend at the same time. It looks like you can use a dictionary to sort days of the week with python	We added a create functionality to all of our pages. All pages now have read and create functionality. The Create form is currently being implemented in the same template as the read functionality, rather than having to redirect to a form page. This has been completed for Ingredients page and is currently being worked on for all other pages. menuItemIngredients page was added for the Menu Item Ingredients intersection table with Read and Create functionality. Update and Delete functionality is being worked on, being completed for the Ingredients page only for now.
Project Step 4	In the outline, you have MenuItems.price as an int, whereas you have it as a Decimal in your DDL. All other datatypes match up. The only other discrepancies are that you do not specify a width parameter (e.g. varchar(255)) in your database outline. For menuItemIngredients, there is no unique identifier for a row, either a unique constraint on the	We've looked to update our documentation, schema, and DDL queries to address the feedback received as part of Project Step 4. The query fixes will be included into our website during the next major update cycle as we've gotten our site stable enough for visiting for step 5 draft submission.

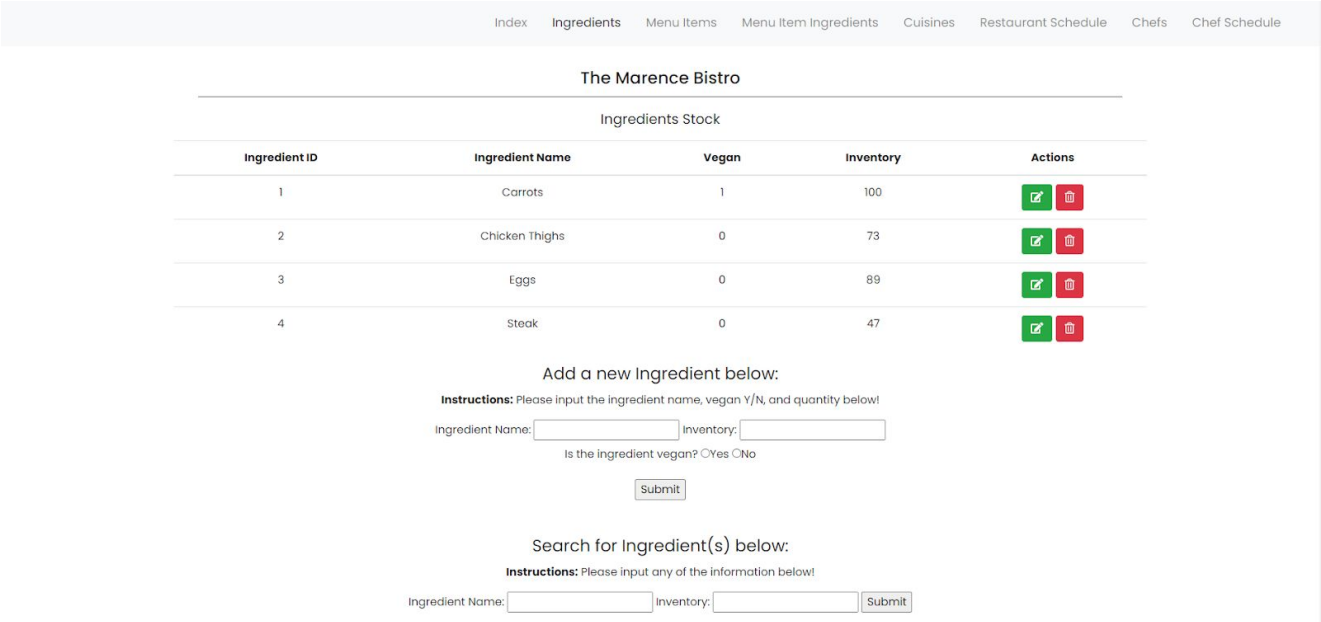
	menuItemID, ingredientID combination, or a composite PK FK of menuItemID and ingredientID, and this can lead to duplicate rows. For 2NF normalization best practices, rows should not be duplicated. There is a discrepancy: the ERD needs to be updated to show the optional relationships between cuisines, chefs and cuisines, restaurantSchedule.	We've fixed discrepancies between documentation and implementation. We've updated our schema to reflect latest implementation of optional relationships We've ensured we've got a primary key (composite or not) for every table.
Project Step 3	"It is difficult to comment more on the design of tables as they do not appear to be implemented yet, but I find the existing layout to work rather well."	We added tables to represent each of the entities in our ERD along with buttons to represent the functionalities for 'Add', 'Delete', and 'Update'.
	"There is no evidence to suggest that any one relationship is NULLable in the front-end due to the lack of tables, but it can be seen in the database outline that for 'ChefSchedule' the chefID fk can be NULLable."	The chefID is a nullable relationship under the ChefSchedule table as well as the cuisineID under the RestaurantSchedule table. e.g. There may be holidays that won't have cuisines or chefs scheduled.
	The addition of search functionality would be a nice addition though this could already be in the works.	Not yet implemented, but we'll be looking to work in search and filtering functionality as we build out the back-end to our website.
DB Entities	You conflated your intersection tables into the entities themselves.	See below
1:M relationships	You should not be including an attribute from another entity that is not the PK of the other entity. For example, menuitemName is included under cuisines when only menuitemID (MenuItems' PK) should be used outside of the MenuItems entity. The same thing goes for including ingredientName as an attribute for the MenuItems entity	Removed all non-PK foreign attributes from tables. Updates reflected as strikethrough and in red text.
M:M relationships	You should use intersection tables to map out the M:M relationships between entities. For example, you have a M:M relationship between MenuItems and Ingredients that should be set up with a table that looks similar to this: MenuItemsIngredients: records the relationship between ingredients and MenuItems - menuitemID: int, not NULL, FK - ingredientID: int, not NULL FK The way you currently have it set up with ingredientID included as a FK under MenuItems would require separate entries into MenuItems for each ingredient that a menu item uses.	Set up intersection tables to help manage our M:M relationships for the following relationships: - Ingredients to Menu Items - RestaurantSchedule to Chefs

UI Screenshots

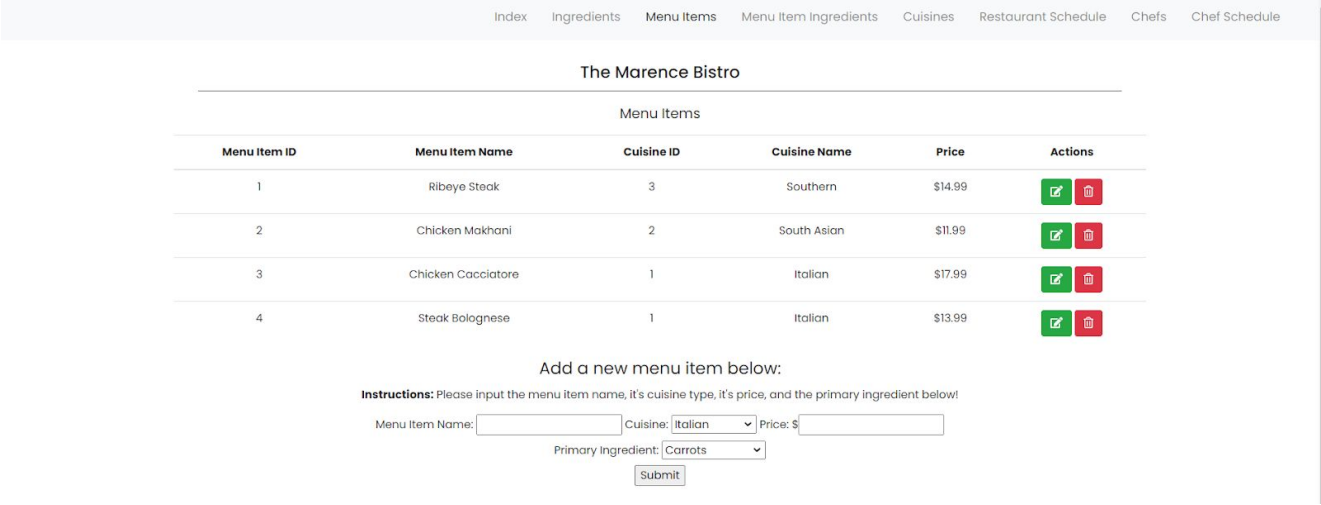
Index Page: Weekly Schedule to browse all Chef shifts for each day with Reset and Search functionality.



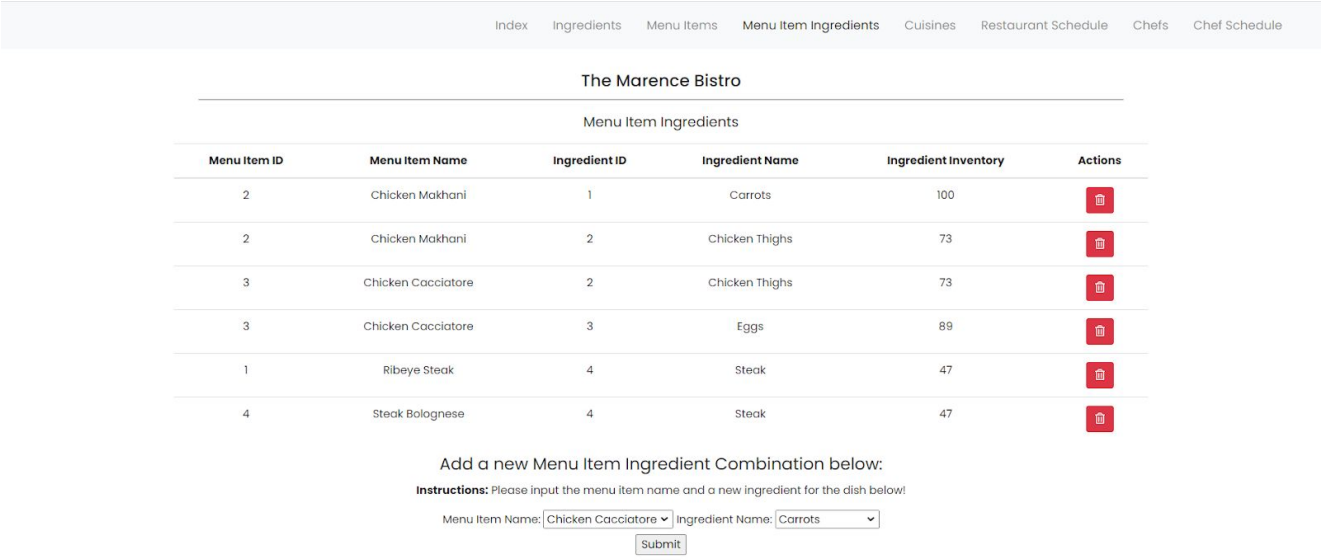
Ingredients Page: Create/Read/Update/Delete and Search functionalities for Ingredients.



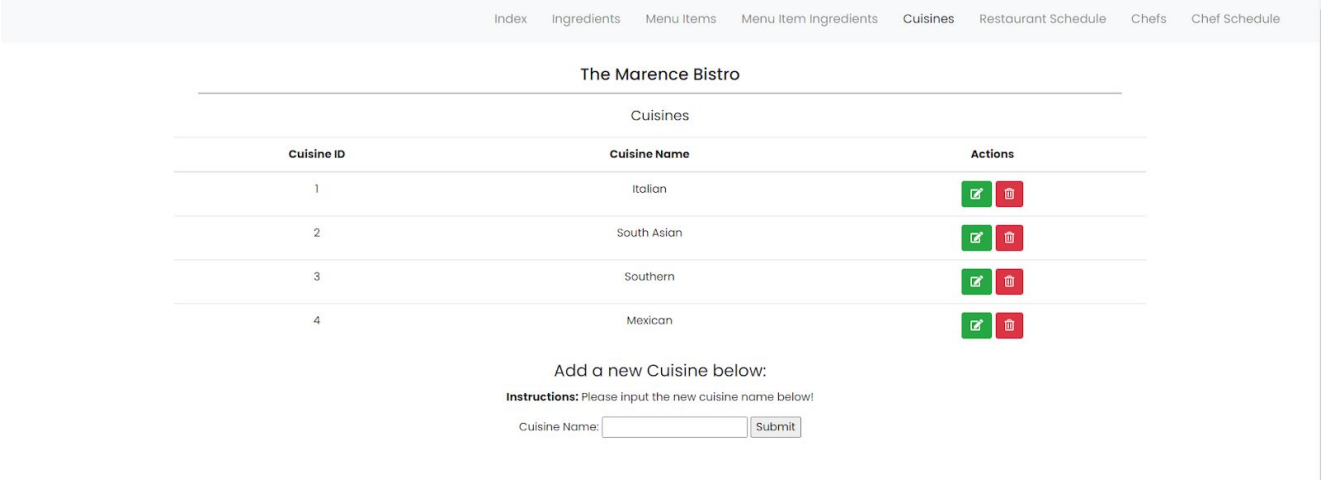
Menu Items Page: Create/Read/Update/Delete functionalities for Menu Items.



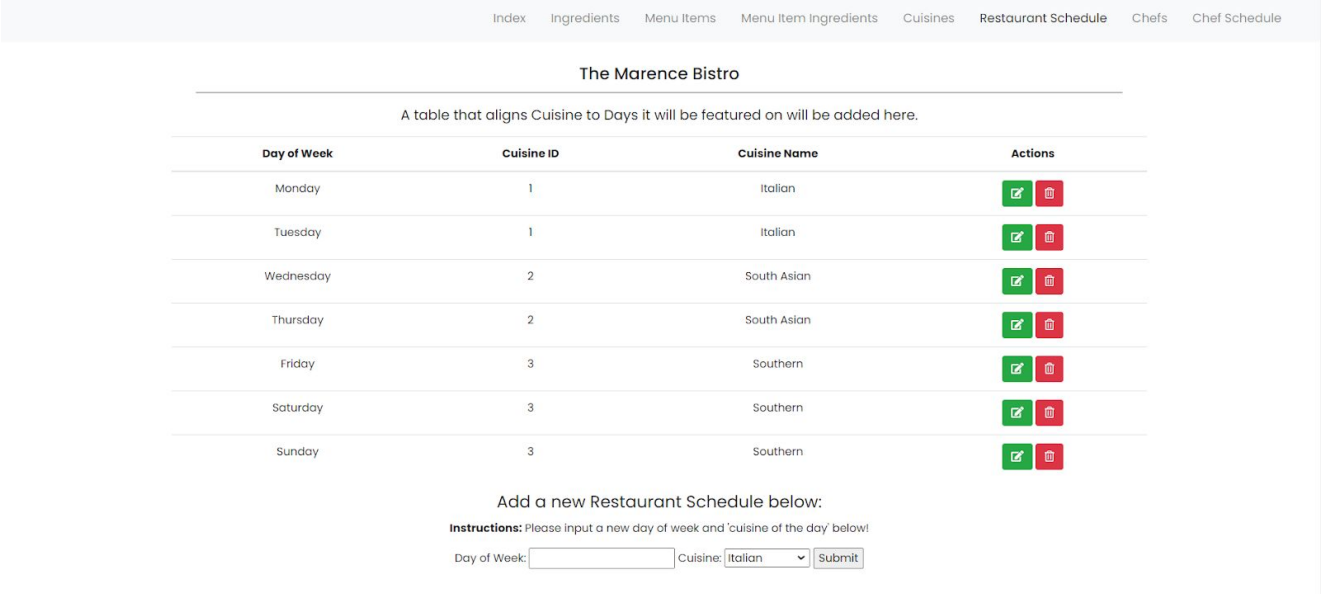
Menu Item Ingredients Page: Create/Read/Delete functionalities for Menu Item Ingredients.



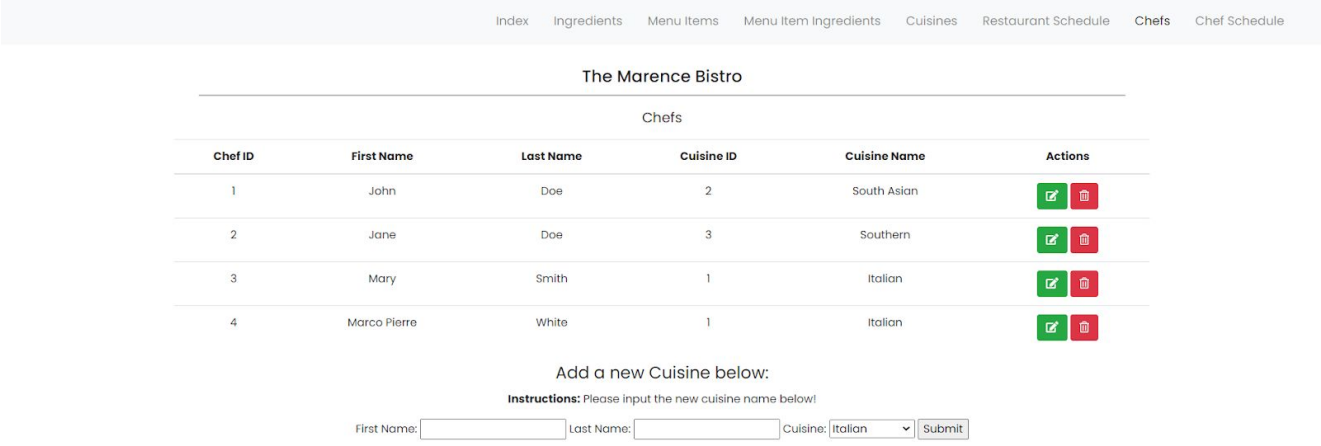
Cuisines Page: Create/Read/Update/Delete functionalities for Cuisines.



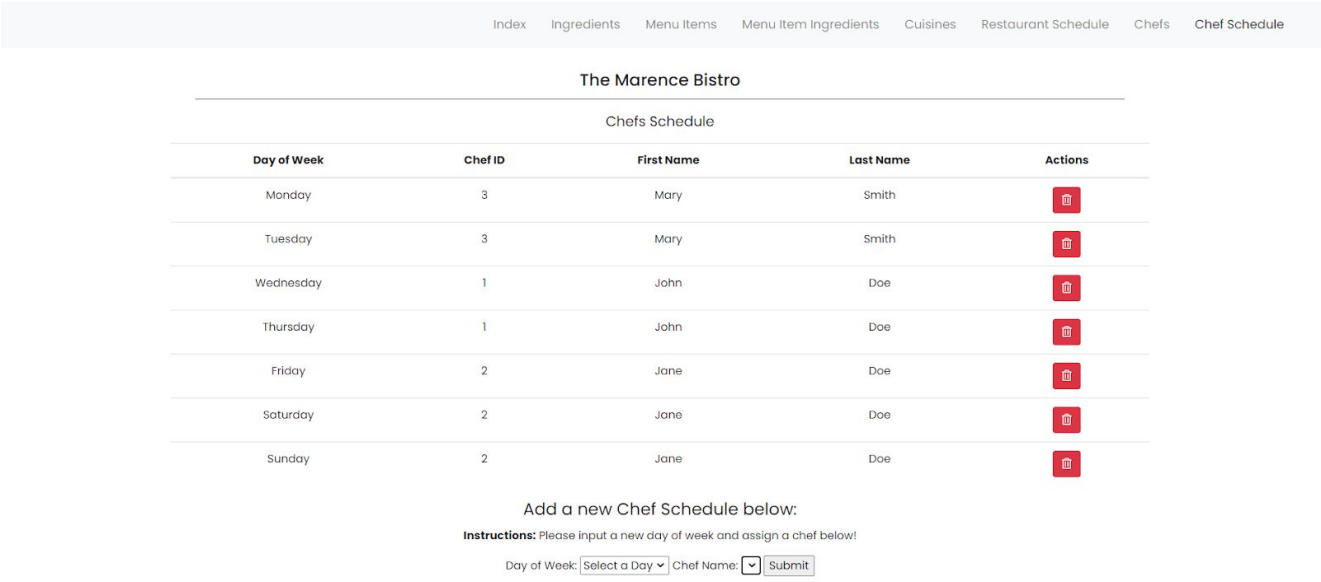
Restaurant Schedule Page: Create/Read/Update/Delete functionalities for Restaurant Schedule.



Chefs Page: Create/Read/Update/Delete functionalities for Chefs.



Chefs Schedule Page: Create/Read/Delete functionalities for Chefs Schedule.



CS 340 Team Evaluation Form - 12/5/2020

Rate your **team's performance** using the scale below.

1 = Strongly Disagree 2 = Disagree 3 = Agree 4 = Strongly Agree

Group number	69	
Name of Group TEAM Members:	Mazen Abdullah, Terence Tang	
SCALE AND COMMENTS	RATING	ADDITIONAL COMMENTS
How prepared was your team? Research, reading, and assignment complete	5	Our team was well prepared for the project. A few hiccups slowed down the progress along the way but the team was able to push through all obstacles and have a satisfactory final piece before the deadline.
How RESPONSIVE & COMMUNICATIVE were you both as a team? Responded to requests and assignment modifications needed. Initiated and responded appropriately via email, Slack etc.	5	Communication was great. Super fast responses and always reachable.
Did both group members Participate equally Contributed best academic ability	5	Both members participated equally in the project as a whole. Mazen was more involved in the front end work while Terence was more involved in the backend work, although both members completed a fair share of work on both ends.

		Overall, both members pulled their weight and achieved a final piece of work that they are satisfied with.
DID YOU BOTH FOLLOW THE initial team contract? Were both team members both positive and productive?	4	For the most part yes. Our initial team contract was meant to be flexible and we've been flexible on who owned which responsibilities. Both team members were positive and productive.

Are there any suggestions for improvement for your team and what are your goals moving forward?

(Better communication, follow the contract better, modify the initial team contract, more contribution, etc?)?

We believe we really hit the nail on this project. The flexibility of our contract helped bring our best work forward at times that work for both of us.