Database Design

Kailee Stenseng

John E. Simon School of Business, Maryville University

SWDV 691: Software Development Capstone

Professor Joseph Gradecki

March 23, 2025

Database Design

For my capstone, I will be utilizing MongoDB and creating a database called niibish-aki. For starters, I am certified in MongoDB, so I am more confident using it than other database types. Secondly, my capstone is document-oriented, and much of my data can vary in structure, such as tea orders and customizations, so utilizing BSON objects would be a better fit. MongoDB is also very scalable, supports references between collections, so I do not need to use JOIN operations to link users to orders, for example.

The web application itself, Niibish Aki, will utilize NextJS's API routes functionality to serve as a REST API layer, allowing the user interface to interact with the database. As discussed in my service layers assignment, all data interactions in Niibish Aki will be routed through this API to ensure a clean separation of concerns between the frontend and backend. Since MongoDB is a NoSQL database, which can handle dynamic schemas, implementing a repository layer is not necessary.

Users

The collection name will be Users. These are for anyone who is signed up on Niibish Aki, and does not mean they have placed an order. This collection is necessary as it stores user's information, such as name, email, their hashed password, address, and favorite items. Although Niibish Aki is designed as a "pick up and pay in person" sort of store, the address information will be used as an identifier for potential abusers of the system, such as people who place orders and do not pick them up. Favorites will be customized beverages saved for quick reordering, and are similar to the Menu items, but without description or category. To prevent the favorites from becoming

unbounded, which is an unrealistic issue, but could theoretically exist, favorites will be limited to 20.

Document Structure

```
[
  {
    “_id”: “ObjectId”,
    “firstName”: “string”,
    “lastName”: “string”,
    “userName”: “string”,
    “password”: “string”,
    “email”: “string”,
    “phone”: “string”,
    “address”: {
      “line1”: “string”,
      “line2”: “string”,
      “city”: “string”,
      “state”: “string”,
      “zipcode”: “string”
    }
  }
]
```

Favorites

The Favorites collection will be storing all users’ favorited beverages. It will reference the User and Menu collections. This might work in the user’s collection, although it runs a risk of becoming unbounded without intervention. So, it will be in its own collection and will make it easier to clear out removed menu items in a post-MVP feature to edit the menu as an admin. Add-Ins will be embedded rather than referenced here as there is no functionality to edit favorited drinks in MVP and there are no stretch goals regarding it.

Document Storage

```
[
  {
    “_id”: “ObjectId”,
    “menuId”: “ObjectId”, //Key Identifier for Menu collection
    “userId”: “ObjectId”, //Key Identifier for User collection
    “name”: “string”,
    “price”: 5.50,
    “addIns”: [
     {
        “name”: “string”,
        “price”: 1.00,
        “amount”: 1
      }
    ]
  }
]
```

Add-Ins

The Add-Ins collection will be for customizations that can be made to a tea, such as popping pearls, boba, or toppings. This collection will be used to support the customization for tea drinks.

Document Structure

```
[
  {
    “_id”: “ObjectId”,
    “name”: “string”,
    “description”: “string”,
    “price”: 1.00,
    “amount”: 1
  }
]
```

Menu

The Menu collection will be for available teas and snacks that will be displayed on the menu page. It stores basic information such as name, description, category, and price, which are all piece of information needed for the product card.

Document Structure

```
[
  {
    “_id”: “ObjectId”,
    “name”: “string”,
    “description”: “string”,
    “category”: “enum”,
    “price”: 5.50,
    “reference”: “string”,
    “alt”: “string”,
    “image”: “string”,
  }
]
```

Order

The Order collection will host all order history. Carts will have a “cart” type, orders that are in progress will have an “in-progress” type, and completed will be “completed”. Again, to prevent the unrealistic chance of these documents becoming unbounded, orders will only be able to hold 20 items. Additionally, the AddIns will be embedded once the order is completed, but referenced until then as to allow modifications in the cart.

Document Structure

```
[
  {
    “_id”: “ObjectId”,
    “userId”: “ObjectId”,        //Key Identifier for User collection
    “items”: [
      {
        “type”: “string”,
        “itemId”: “ObjectId”,    //Key Identifier for Menu collection
          “addIns”: [
            {
             “_id”: “ObjectId”,  //Key Identifier for AddIn collection
             “name”: “string”,
```

```
            “price”: 1.00,
            “amount”: 1
          }
        ],
        “quantity”: 1,
        “price”: 5.50
      }
    ],
    “bill”: {
      “subtotal”: 10.00,
      “tax”: 1.00,
      “tip”: 1.00,
      “total”: 12.00
    },
    “status”: “string”,
    “createdAt”: “ISODate”
  }
]
```

Indexes

Indexes will be used to enhance performance with frequent lookups. Indexes will be created on the Users collection by `email` for the login. They will also be used on the Orders collection for `userId` to get order history, `status` for filtering carts, and `createdAt` to sort user history. The Menu collection will have indexes for filtering food and tea by `category`, and for getting featured items by `category`. The Favorites collection will have indexes for fetching user’s favorites by `_id` and for cleaning up favorites when item is removed with the `menuId`.