CS3106 - Practical 2

200007413

Word Count: 1745

1 Compilation, Execution & Usage Instructions

1.1 Technologies Used

This submission uses HTML, CSS, javascript, and Node.js. Vue.js was used for the client, and Express.js with Knex was used for the server.

1.2 Requirements

- npm (tested on 9.6.2)
- node (tested on v18.15.0)
- mariadb (tested on v15.1 distrib 10.5.18-MariaDB, for Linux x86_64)

1.3 Setup

1. Make sure you have a MariaDB user and database. Then, import the dump
   ```
   mariadb -p DATABASE_NAME < gp87_cs3101_p2.sql
   ```
2. Set the server environmental variables in `src/server/.env`.
3. Set the client environmental variables in `src/client/.env`.
4. Install server dependencies. Navigate to `src/server` and run:
   ```
   npm install
   ```
5. Install client dependencies. Navigate to `src/client` and run:
   ```
   npm install
   ```

1.4 Running

1. Navigate to `src/server` and run:
   ```
   npm start
   ```
2. In another terminal, navigate to `src/client` and run:
   ```
   npm run dev
   ```

1.5 Troubleshooting

- Client `ECONNREFUSED`.

  (a) The server is not running (see. subsection 1.4)

  (b) Environmental variables are not correct (see. subsection 1.3).

- Server on startup: `[nodemon] app crashed`

  MariaDB environmental variables are not correct or database user does not have adequate permissions (see. subsection 1.3).

2 Overview

Overall, the database implementation fully represents the schema provided in the specification with strict attribute type checking, validations, and a considered compromise between size and flexibility. Restraints on updating, and deleting are enforced where appropriate to maintain priority data.

Each of the required SQL queries are implemented successfully, and have their own views. The required stored procedures are also all implemented, with human-readable error messages when the input is invalid or entries cannot be inserted.

The GUI allows organisers to see information about guests, and change the arranged table numbers. Notes can be set for each guest, which are only viewable in the organiser section. Organisers can add new guests to an existing invitation and table.

The GUI also enables guests to use their invitation code to access information about guests associated with that code. They are allowed to edit responses for these guests, and set dietary requirements, or add new ones to the database.

A login system protects organiser information by requiring an organiser ID and password. To prevent malicious API calls, the server checks the session status for organisers, and requires guests to provide the invitation code for every request which is checked against the guest being edited.

Errors are shown to the user when an interaction fails. Where the input is invalid, failed validation, or database constraints prevent interactions, relevant error messsages are displayed. Where system errors not caused by user input occur, a generic system error message is displayed.

3 Database Implementation

The dietary_requirement table has two VARCHAR columns for the name and description of each requirement. It was decided that short_name should have a limit of 20 characters to reduce the size of the table, as additional information can be stored in the longer 128 character description. As the short_name is the primary key, it cannot be null, but the description can be.

The dinner_table table has two integer columns for the table_no and capacity. The table_no uses the TINYINT UNSIGNED data type, as it was assumed that tables will have only positive numbers, and there is unlikely to be more than 255 tables. The capacity is of type MEDIUMINT UNSIGNED, to allow for many quests to be assigned to a table, but this ensures that the capacity is always greater than 0 as per the specification.

The invitation table contains two VARCHAR columns for the code, and address. The primary key attribute 'code' is limited to 10 characters, which has a very large combination of unique values, allowing many invitations to be created. The address contains 245 characters, as the UK government has recommended international postal addresses have a maximum of 7 lines of 35 characters for a full address. The other column is date_sent, which has the DATE type, and a trigger when updating or inserting rejects values that are in the past (current date determined by system) as per the specification.[1]

The person table has the id primary key is of type INT UNSIGNED, as this allows many guests and organisers to be added, and it was decided that no benefit came from allowing negative id values. The full_name VARCHAR has a maximum size of 70 characters, which was recommended by the UK government,[1] and cannot be null to help identify the person. The response attribute is a BOOLEAN type, but can be null, to help keep track of guests who haven't responded either way yet.

There are two foreign key columns in the person table for invitation_code and table_no. Both can not be null, to make sure guests are all accounted for in the tables and invites. When an invitation code is deleted, the people associated are deleted, which allows for batch removal of guests. If a dinner_table entry is deleted but is referenced by a person, then the deletion is restricted to prevent accidental removal of people. Both foreign keys update the reference when their primary key value changes.

To enforce the dinner_table capacity, when a person is updated or inserted, a trigger is executed before, which counts the person entries (excluding the one being inserted or updated) referencing the given table_no, and makes sure the capacity is not exceeded.

The guest_diet table has two foreign keys linking people with dietary requirements. To aid with moderation, when a dietary_requirement is removed from the database, so are the guest_diets referencing it. Similarly, the guest_diet cascades the deletion of the referenced person, as the person no longer exists in the database, so should have no relationships.

4 GUI Implementation

4.1 File Summary

```
src
├─ server
│  └─ src
│     ├─ db
│     │  └─ db.js
│     └─ index.js
└─ client
   ├─ src
   │  ├─ assets
   │  │  ├─ base.css (basic css styling generated by vue.js)
   │  │  └─ main.css (basic css styling generated by vue,js)
   │  ├─ components
   │  │  └─ GuestsTable.vue
   │  ├─ router
   │  │  └─ index.js
   │  ├─ stores
   │  │  ├─ guest.js
   │  │  └─ organiser.js
   │  ├─ views
   │  │  ├─ GuestsLogin.vue
   │  │  ├─ GuestsView.vue
   │  │  ├─ OrganisersLogin.vue
   │  │  └─ OrganisersView.vue
   │  ├─ main.js
   │  └─ App.vue
   ├─ index.html:  (basic html template generated by vue.js)
   ├─ vite.config.js:  (vue networking configuration generated by vue.js)
   └─ vitest.config.js:  (testing configuration generated by vue,js)
```

Figure 1: Forest diagram displaying the relevant files to the GUI operation, with notes where files were generated automatically.

4.2 Server

There are two parts to the GUI implementation. First, the server folder, which contains an Express.JS node server in `index.js`. The separation of client and server helps keep the database secure from users, as only the data the server sends is accessible.

The express app has post and get endpoints which are either intended for guests or organisers to use.

For the guest endpoints where guest info is retrieved or updated, the request body must contain an invitation_code value. This then determines the user information sent back, and is cross-referenced when guest responses or dietary requirements are updated with the relevant guest invitation code from the database.

The organiser endpoints validate that the organiser is logged in. For this, the endpoint `/api/auth/organisers` accepts an organiser_id and password, which is verified using the database, and then creates a session, which is referenced by a cookie in the client's browser. This means that a password is not needed for every call to the database. To prevent session hijacking,[2], sessions last 15 minutes before expiring - at which time a new login is required.

The `db.js` module in the server section contains the methods which directly interact with the database. This allows a conceptual distinction between the request parsing and validation occurring in the endpoint functions, and the usage of those values to call the database.

Here, Knex.js was used as it makes creating prepared statements in JavaScript easy. This enhances the security of the server, as Knex.js is widely used, reducing the reliance on my own personal input validation to prevent SQL Injection attacks.[3] The only command which uses non-prepared statements is where the aggregation of dietary requirements into a JSON array is used, as Knex does not provide a function for this. However, statement uses no user input, which means the risk of injection attacks is already prevented.

4.3 Client

The other part of the GUI allows guests and organisers to see and interact with data from the database by calling the server.

The client is a single page web-app created using the Vue.JS framework. The files `index.html` and `main.js` are used to generate the app. `App.vue` contains the default app layout, displaying a navigation bar and the current page.

The `router.js` file is a Vue-Router configuration which defines the page destinations and views used for each page. When a path is resolved, the `beforeEach` function in this file runs, which is used to redirect guests and organisers to the relevant login pages if not authenticated.

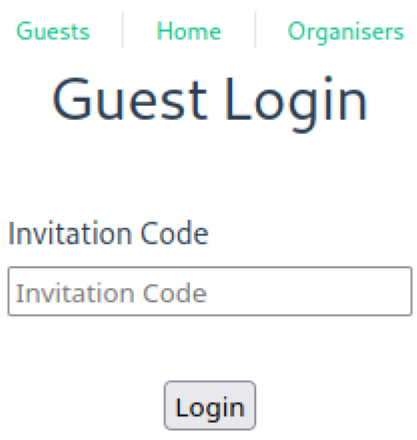


Figure 2: Guest login screen which is displayed before a guest authenticates using their invitation code

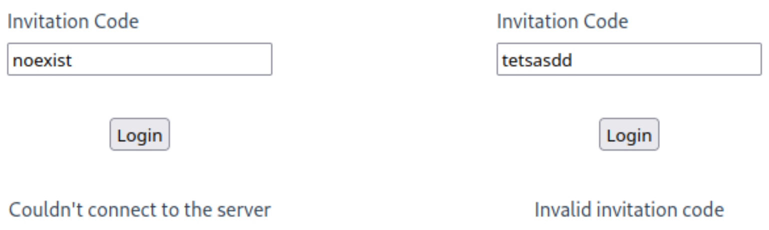
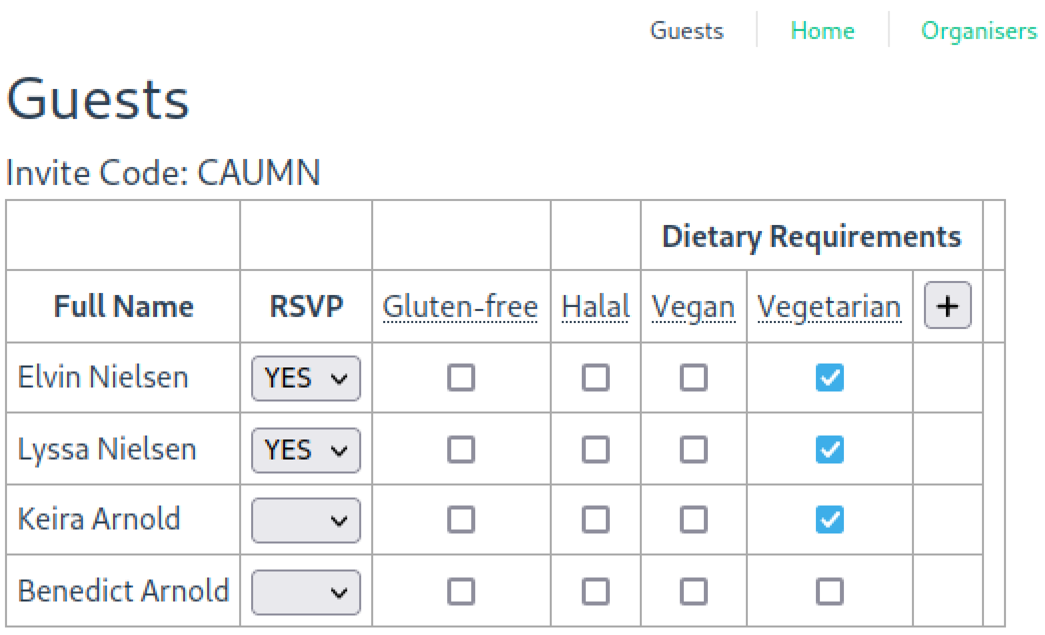


Figure 3: Error message displayed when the submitted guest invitation code is not in the database

The guest section is accessed through the header. If not yet authorized, they are presented with an invitation input form (see. Figure 2). On submit, the form calls the `getGuests` method in the `stores/user.js` store, which stores the returned list of guests associated with the code.

Figure 3 shows relevant error messages are shown below the form giving information about the reason why the request failed, so the guest knows if they have made a mistake or not.

When a valid invitation code is presented, the list of associated guests are retrieved from the database and temporarily stored in the guest store, along with the invitation_code for authorization if changes are made.



Guests

Invite Code: CAUMN

				Dietary Requirements		
Full Name	RSVP	Gluten-free	Halal	Vegan	Vegetarian	+
Elvin Nielsen	YES	☐	☐	☐	✓	
Lyssa Nielsen	YES	☐	☐	☐	✓	
Keira Arnold		☐	☐	☐	✓	
Benedict Arnold		☐	☐	☐	☐	

Figure 4: Table of guests displayed after a valid invitation code has been presented in the guests section

A table defined in `components/GuestTable.vue` is then presented, which displays the relevant information, which can be seen in Figure 4. This implementation uses a table, as it is an intuitive medium for displaying lots of similar pieces of information.

Guests can RSVP using a drop-down, where the blank response represents no response. There are checkboxes allowing guests to toggle existing dietary requirements in the database. Hovering over a requirement name displays it's description if it exists. A button in the dietary requirements section allows guests to add new requirements, which presents a pop-up input for the name and description (see. Figure 5).

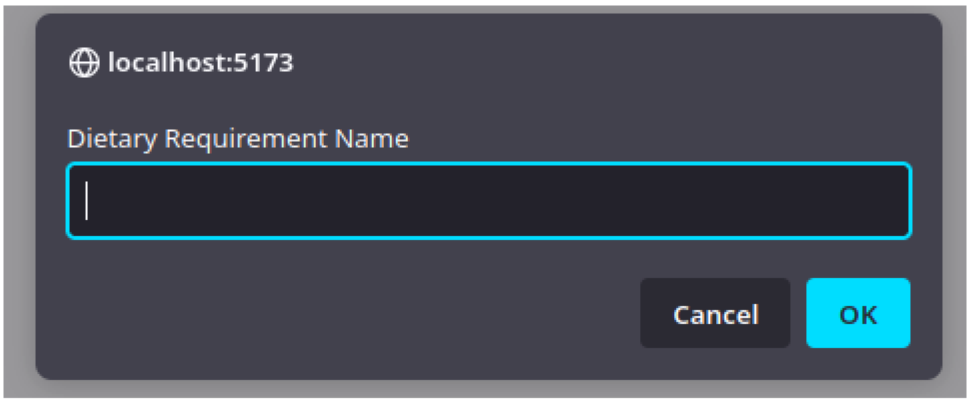
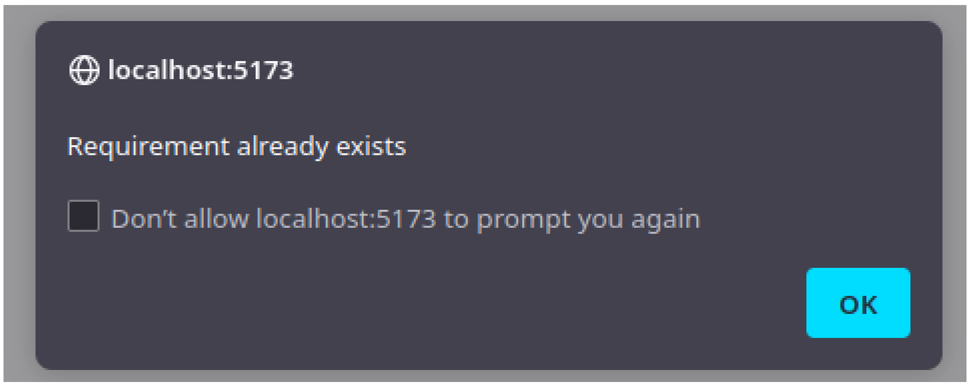


Figure 5: Pop-up displayed to users when they choose to add a new dietary requirement


localhost:5173

Requirement already exists

Don't allow localhost:5173 to prompt you again

Figure 6: Error presented when the dietary requirement is already in the database

When the request fails, a relevant error is displayed using a popup alert. Here, the values in the table are reverted back to maintain consistency with the database (see. Figure 6).

When first navigating to the organiser section, users are redirected to the organiser login page defined in `views/OrganiserLogin.vue` (see. Figure 7). Similar error messages are displayed to the guest login section.

Once authenticated, the guest information is shown in the same table defined in the `components/GuestTable.vue`, but more columns are shown. This is shown in Figure 8. From here, they can edit the table numbers from existing tables, delete guests, add dietary requirements, or add a guest by filling in the form which is shown as the first row of data in the table. Figure 9 shows that, upon trying to update or insert a guest where the selected table is already full, a relevant error is displayed.

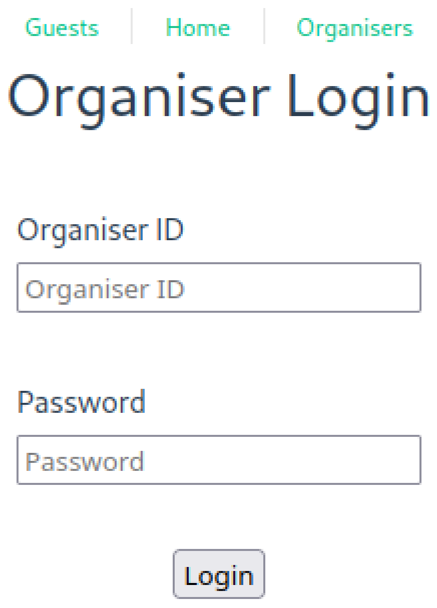
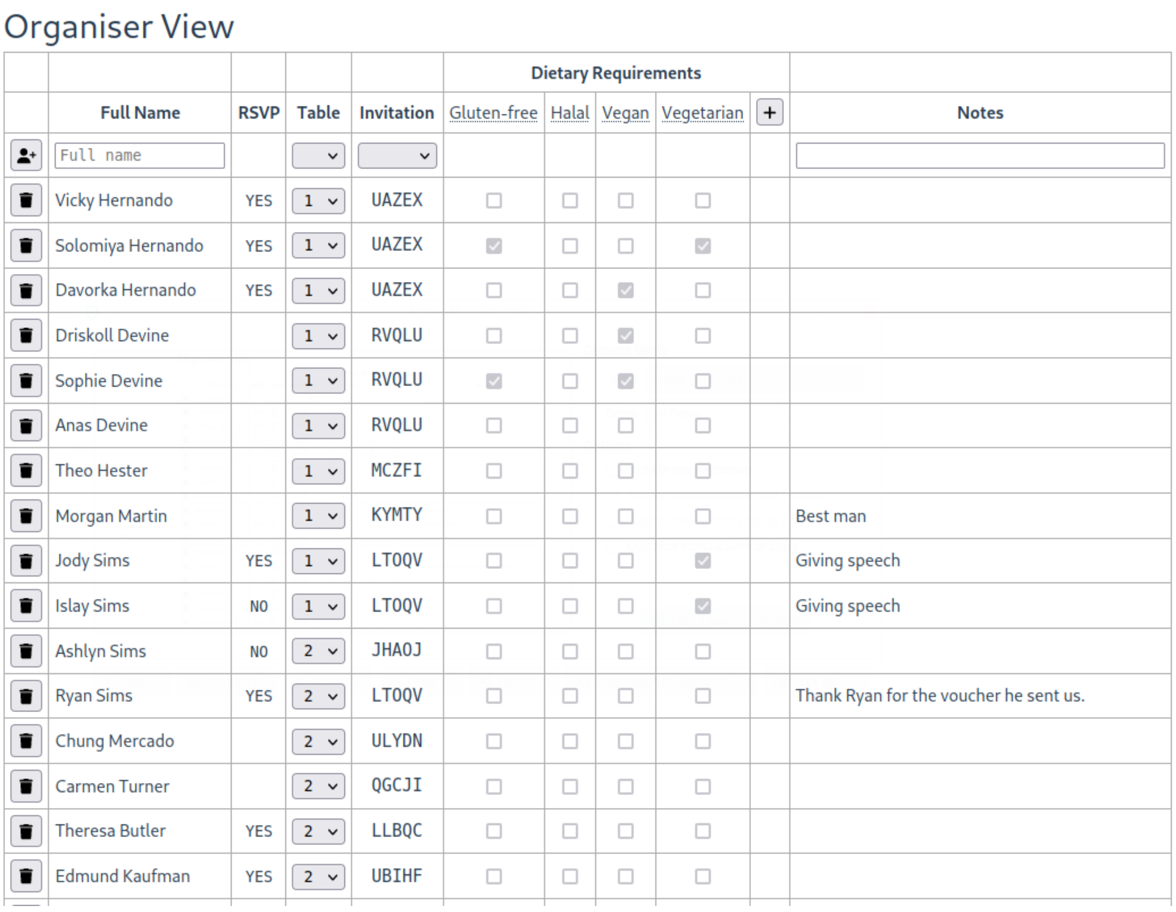


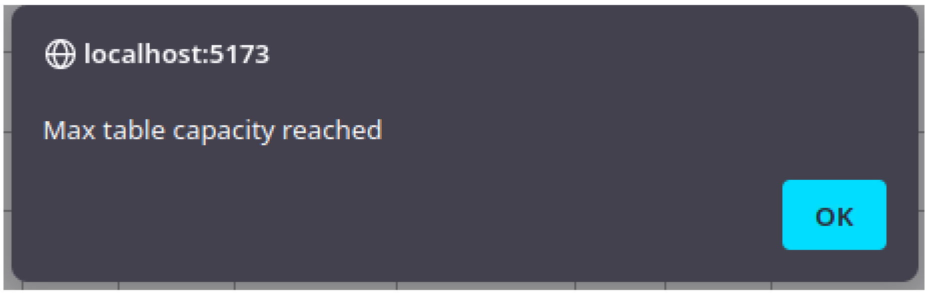
Figure 7: The organiser login screen



Organiser View

					Dietary Requirements					
	Full Name	RSVP	Table	Invitation	Gluten-free	Halal	Vegan	Vegetarian	+	Notes
	Full name									
	Vicky Hernando	YES	1	UAZEX	☐	☐	☐	☐		
	Solomiya Hernando	YES	1	UAZEX	☑	☐	☐	☑		
	Davorka Hernando	YES	1	UAZEX	☐	☐	☑	☐		
	Driskoll Devine		1	RVQLU	☐	☐	☑	☐		
	Sophie Devine		1	RVQLU	☑	☐	☑	☐		
	Anas Devine		1	RVQLU	☐	☐	☐	☐		
	Theo Hester		1	MCZFI	☐	☐	☐	☐		
	Morgan Martin		1	KYMTY	☐	☐	☐	☐		Best man
	Jody Sims	YES	1	LTOQV	☐	☐	☐	☑		Giving speech
	Islay Sims	NO	1	LTOQV	☐	☐	☐	☑		Giving speech
	Ashlyn Sims	NO	2	JHAOJ	☐	☐	☐	☐		
	Ryan Sims	YES	2	LTOQV	☐	☐	☐	☐		Thank Ryan for the voucher he sent us.
	Chung Mercado		2	ULYDN	☐	☐	☐	☐		
	Carmen Turner		2	QGCJI	☐	☐	☐	☐		
	Theresa Butler	YES	2	LLBQC	☐	☐	☐	☐		
	Edmund Kaufman	YES	2	UBIHF	☐	☐	☐	☐		

Figure 8: The interactive table shown to organisers once logged in



localhost:5173

Max table capacity reached

Figure 9: Error message displayed when organisers try to insert a guest or update a guest where the table is already full.

References

[1] e-Government Interoperability Framework (e-GIF). *Government Data Standards Catalogue Volume 2 - Data Types Standards*. 2001. URL: `https://webarchive.nationalarchives.gov.uk/ukgwa/+/http://www.cabinetoffice.gov.uk/media/254290/GDS%20Catalogue%20Vol%202.pdf` (visited on 04/05/2023).

[2] Vineeta Jain, Divya Rishi Sahu, and Deepak Singh Tomar. “Session hijacking: threat analysis and countermeasures”. In: *Int. Conf. on Futuristic Trends in Computational Analysis and Knowledge Management*. 2015.

[3] Anh Nguyen-Tuong et al. “Automatically hardening web applications using precise tainting”. In: *Security and Privacy in the Age of Ubiquitous Computing: IFIP TC11 20 th International Information Security Conference May 30–June 1, 2005, Chiba, Japan 20*. Springer. 2005, pp. 295–307.