Review

The project layout looks as follows

/ `CodeInfinityInternApplication`

```
├── / Resources
│   ├── / php
│   │   ├── check_uniqueness.php # Ensure no duplicate lines exist
│   │   ├── create_member.php # Add a new MongoDB member
│   │   ├── generate_file.php # Generate random data for CSV
│   │   ├── load_sqlite.php # Load data from CSV to SQLite
│   │   ├── manage_mango.php # Handles MongoDB connections/logic
│   │   ├── manage_sqlite.php # Handles SQLite DB operations
│   │   ├── read_database.php # Read entries from SQLite
│   │   ├── read_file.php # Read entries from CSV file
│   │   ├── upload_file.php # Upload and process user CSV files
│   │   └──
│   ├── / css
│   │   ├── generate.css # Styles for generate.php
│   │   ├── home.css # Styles for index/home page
│   │   ├── loading.css # Styles for loading screen
│   │   ├── member.css # Styles for member form
│   │   └──
│   ├── / database
│   │   ├── csv_database.sqlite # SQLite database file
│   │   └──
├── / output
│   ├── output.csv # Generated or imported CSV data
├── / uploads
│   ├── test.csv
│   └──
├── database.php # Show SQLite database in table format
├── generate.php # Form to generate and view CSV data
├── import.php # Form to import/upload CSV file
├── index.php # Home/landing page
├── loading.php # Handles background data loading
├── member.php # Form to add new members (MongoDB)
├── view.php # View MongoDB members table
```

Explanation

1. `Check Uniqueness`

- I made use of dictionaries to check whether the line has occurred before
- If it did, I just generated a new line in its place

2. `Create Member`

- Checked ID length
- Checked for invalid characters with name and surname
- Check that the dates are correct
- Made use of MangoDB to handle data

3. `Generate File`

- I put the 20 names and surnames into 2 different arrays and then used a random number to get the index of the random name. Same with the age. I then chose a random day out of 365 and subtracted the age and the random day with the current date to get date of birth

4. `Load SQLite`

- First remove everything from the file.
- Then Initialize the database and create the table over again
- Then add the certain amount of entries

5. `Manage Mango`

- Just two functions to get the members between `min` and `max`
- And also to get the size of the collection

6. `Manage SQLite`

- Just functions to handle the SQLite database

```
# Function to initialize database
function init(): void

# Function to print the rows from the database between $min and $max
function print_rows($min, $max): void

# Function to get the size of the table
function size(): void

# Function to remove all the contents from the database
function remove_file_sql(): void

# Function to insert rows into the table/database
function insert_entry($id, $name, $surname, $intials, $age, $dob):
```

```
bool
```

7. `Read Database`

- I was going to implement something here but just did it in the `Manage Mango` file

8. `Read File`

- Checks if the file needs to print the header or body
- And then `echo` the rows as requested by the `index` variable

9. `Upload File`

- Loads the files into `output.csv` and also the file into uploads folder

Programming Techniques

1. Using `JQuery` and `AJAX` (Recursion)
2. Require function files to help with `UI`
3. Making use of Indexes to print certain rows
4. SQLite handling
5. Mango Database handling

Using `JQuery` and `AJAX`

Import `JQuery`:

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
```

Use `AJAX`:

```
$.ajax({
    url: link,
    type: "POST",
    data: postData,
    success: function (response) {
        document.querySelector("#container").textContent = index;
        console.log(response);
        index++;
        sendNextBatch();
    },
    error: function (xhr, status, error) {
        alert("There was an error. Please try again later.");
        console.log(error);
```

```
    }
});
```

This code is used to send data via `POST` to the `link` provided and then that `php` file runs and sends a response and that response, if successful, is used to run recursion.

Require function files to help with `UI`

The require function returns all the code from the `url`. So you can `echo` all the html code from that file so that your code is cleaner.

```
    if (isset($_GET['index'])) {
        $_SESSION['index'] = $_GET['index'];
        require("Resources/php/read_file.php");
    }
```

Making use of Indexes

The `$_GET['index']` is used to only print the rows from that index load. Each load consist of 100 rows.

```
if (isset($_GET['index'])) {
    $index = $_GET['index'];
    require("Resources/php/manage_sqlite.php");
    print_rows((($index - 1)*100), ($index*100));
}
```

```
function print_rows($min, $max) {
    $db = new PDO("sqlite:Resources/database/csv_database.sqlite");
    $limit = $max - $min + 1;
    $offset = $min;
    $stmt = $db->query("SELECT * FROM csv_import LIMIT $limit OFFSET
$offset");
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        echo "<tr>";
        foreach ($row as $value) {
            echo "<td>" . htmlspecialchars($value) . "</td>";
        }
        echo "</tr>";
    }
}
```

SQL Handling

In regards to the indexing. You can't say that it needs to return from the min to the max. Instead you can limit the amount of rows returned and then create an offset from

where you should start returning.
Shown in Making use of Indexes