Assignment 2

Thinks you will learn □

In this assignment, you will learn the following:

- Save your data persistently in MongoDB
- Perform DB different operations
- Deploy your code to a Cloud Provider
- Respond with HTML files and JSON
- Render Dynamic HTML files
- Push to GitLab Repository
- Work in teams

Assignment 2 Theme

Due to a large number of events and categories, Monash's IT department started to think of the need to store their data persistently. Two more issues have emerged: the IT department cannot host the application locally on-premise. Instead, it prefers to deploy it to a cloud provider that charges per usage. Secondly, Monash mobile (Android and IOS) and desktop applications have requested access to the EMA data.

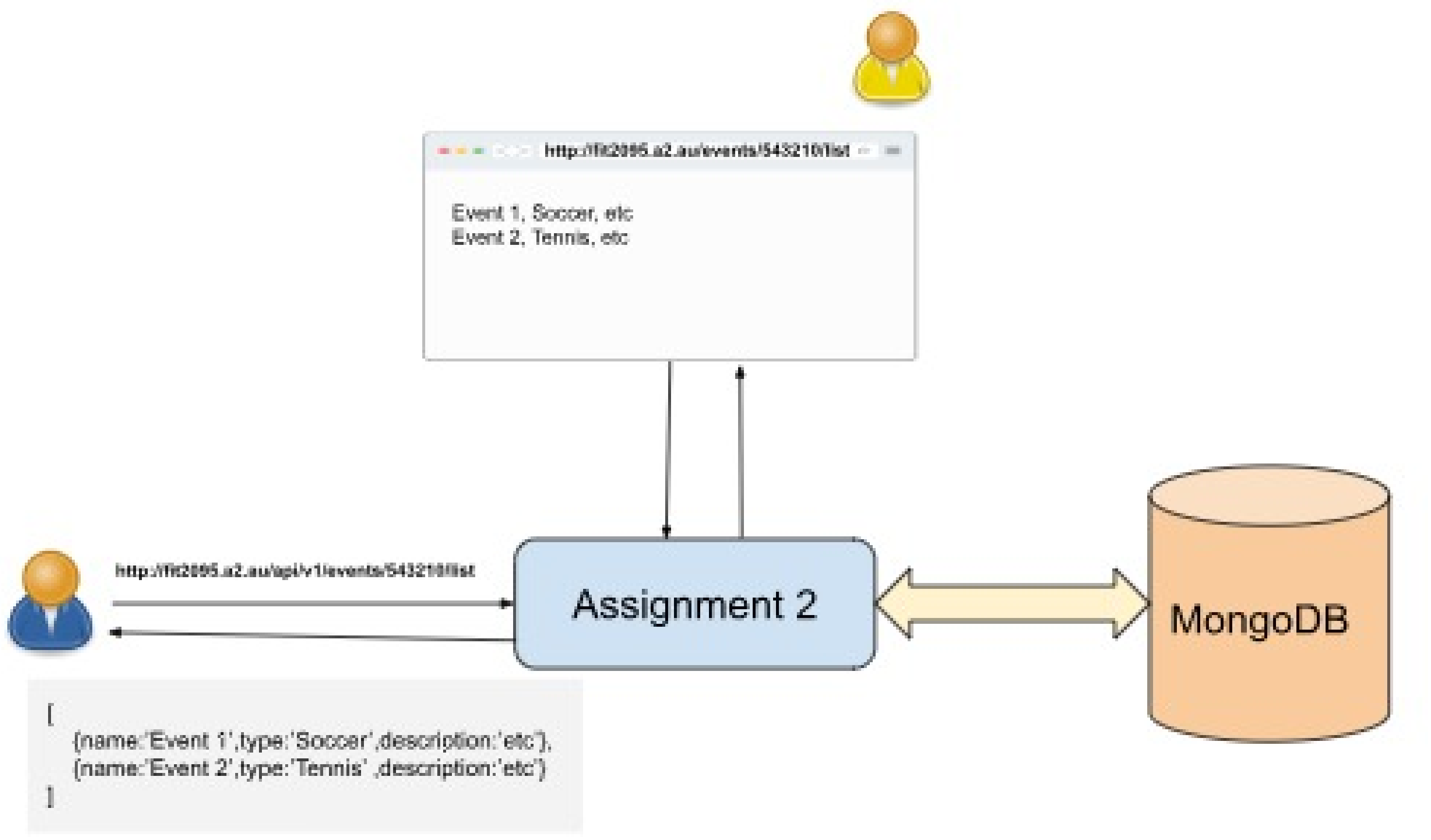
Assignment 2 Specifications ⚙

Assignment 2 is due on Friday of Week 8, Septermber 15, at 11:55 pm.

As mentioned in the Assignment 2 Theme section, this assignment is about:

1. Save EMA data persistently in MongoDB. In other words, you must remove the array of objects you have used in assignment-1 and replace it with MongoDB
2. Serve clients with JSON data
3. Deploy your EMA app to a cloud provider

Assignment 2 Structure


http://fit2095.a2.au/events/543210/list
Event 1, Soccer, etc
Event 2, Tennis, etc
http://fit2095.a2.au/api/v1/events/543210/list
Assignment 2
MongoDB
[
{name:'Event 1',type:'Soccer',description:'etc'},
{name:'Event 2',type:'Tennis' ,description:'etc'}
]

In the figure above, you will notice the following:

1. The EMA project is serving two types of URLs
   1. http://fit2095.a2.au/events/543210/list
      This is developed in assignment 1, where the server responds with HTML files
   2. http://fit2095.a2.au/**api/v1**/events/543210/list

        - where 543210 is the student id,
        - and fit2095.a2.au is the server address
          In the second type, the server responds with JSON data. This type of REST API is designed for other devices, such as mobile phones, TVs, and cars.

There is no direct access to the data. In other words, clients can access the data through the server only.

Teamwork

- Produce together the overall app design/development plan
- Produce together the team tasks
- Agree on **who does which** set of individual tasks
- Check-in regularly with one another
- Anything else about how you'd like them to work together [e.g. git commit authorship, branches, notes from meetings, Trello boards, etc.]

Individual work: You should produce the individual tasks yourself but should discuss ideas with your team

Team Tasks

1. Design the endpoints routing table. In this table, you must list all the endpoints your project will listen to, their methods, and their description. It's your task to name the pathnames, endpoints, and query string parameters.
2. The team is responsible for adding the following features to the index.html file
    1. Two labels that show the current number of events and categories
    2. Three labels show the number of Add, Update, and Delete operations invoked on the database.
       (HINT: This can be implemented by adding a new collection (table) with three counters that are incremented when their operations are invoked.)

Individual-Work Tasks

Each student in the team must pick one group of individual tasks to implement. However, cooperation between the team members is crucial for all the specs and features to work seamlessly.

Tasks Group 1 (student #1)

You MUST use your Monash ID as a segment in the pathname of all the endpoints of this group of tasks. For example, if your Monash ID is 543210, the endpoint could be `http://localhost:8080/category/543210/add`

1. Develop a Mongoose Schema for Category class
    1. Apply 'Required' for mandatory fields
    2. Add a validator to the field 'Name' to accept alphanumeric values only
    3. The category schema must contain a field named '**eventList**', which is an array of events' references. It lists all the events in this category. (HINT: You will use it with the function '**populate**')
2. Add new RESTful API endpoints that are responsible for the following:
    1. Insert new category. The data in a JSON format must be sent through the body of the HTTP request. An example of the endpoint URL could be *http://localhost:8080/api/v1/category/543210/add*

        ```
        {
            "name":"Cat3",
            "description":"Desc"
        }
        ```

        The endpoint must respond with a JSON object containing the category ID of the newly added category
        For example:

        ```
        {
            "id": "CTE-4899"
        }
        ```

    2. List all categories. An example of the endpoint URL could be *http://localhost:8080/api/v1/category/543210/list*
        This endpoint must return the list of all categories and the details for their events in a JSON format (see the populate method)
3. Add new RESTful API endpoints that are responsible for the following:
    1. Delete category by ID. This endpoint deletes a category by its ID and all the events that are listed in the 'eventList' field (see point 1.3). NO further delete cascade is required.
        The ID must be sent through the body of the request in a JSON object. For example

        ```
        {
            "categoryId": "CCA-4960"
        }
        ```

        The endpoint must return a JSON object containing the number of deleted documents. (**Remember to use the correct HTTP method**.)
        Example of the return object:

        ```
        {
        ```

```
    "acknowledged": true,
    "deletedCount": 1
}
```

4. Update category name and description by ID: The ID, new name and description are sent as a JSON object through the request body. For example:

   ```
   {
       "categoryId": "CCA-3001",
       "name": "new Name",
       "description": "new description"
   }
   ```

   The endpoint must return an object with a message to confirm the update or if the ID is not found (HINT: check the return of the update statement). For example:

   ```
   {
       "status": "ID not found"
   }
   ```

5. Update all the assignment-1 endpoints to use retrieve, insert, and delete SQL operations from MongoDB using Mongoose
6. Deploy your project to a virtual machine on your GCP account. The UI and the API endpoints must be accessible through your public IP address.

Tasks Group 2 (student #2)

You MUST use your name as a segment in the pathname of all the endpoints of this group of tasks. For example, if your name is David, the endpoint could be `http://localhost:8080/event/david/add.`

1. Develop a Mongoose Schema for the Event class
   1. Apply 'Required' for mandatory fields
   2. add a validator to the field 'Capacity' to accept numbers between 10 and 2000 (inclusive) only
   3. The event schema must contain a field named '**categoryList**', which is an array of categories' references. It lists all the categories in this event. (HINT: You will use it with the function 'populate')
2. Add new RESTful API endpoints that are responsible for the following:
   1. Insert new event. The data in a JSON format must be sent through the body of the HTTP request. An example of the endpoint URL and its request could be: *http://localhost:8080/studentName/api/v1/add-event*

```
{
  "name": "Event6",
  "description": "Desc ",
  "startDateTime": "2023-08-01",
  "durationInMinutes": 100,
  "capacity": 100,
  "categories": "CCA-3001,CCA-7895"
}
```

The ID of the new event must be added to the 'eventList' array field in all the categories under this event.
The endpoint must return an object containing the event ID of the new event. For example:

```
{
    "eventId": "EKU-1429"
}
```

2. List all events: This endpoint must return the list of all events and the details for their categories in a JSON format (see the populate method)

   An example of the endpoint URL could be: *http://localhost:8080/studentName/api/v1/events*

3. Add new RESTful API endpoints that are responsible for the following:
   1. Delete events by ID. The ID is sent as a JSON object through the body of the request. For example:
      http://localhost:8080/studentName/api/v1/delete-event

      ```
      {
          "eventId":"EPP-5901"
      }
      ```

      The ID of the deleted event must be removed from the 'eventList' array in all the categories that are listed in the 'categoryList'. No further delete cascade is required.
      Example of the return object:

      ```
      {
          "acknowledged": true,
          "deletedCount": 1
      }
      ```

4. Update event name and capacity by ID. The ID, new name and capacity are sent as a JSON object through the request body. For example:
   http://localhost:8080/studentName/api/v1/update-event

   ```
   {
       "eventId":"EKU-1429",
   ```

```
    "name":"new event",
    "capacity":300
}
```

The endpoint must return an object with a message to confirm the update or if the ID is not found (HINT: check the return of the update statement). For example:

```
{
    "status": "updated successfully"
}
```

5. Update all the assignment-1 endpoints to use retrieve, insert, and delete SQL operations from MongoDB using Mongoose. Don't forget, events now can be under multiple categories, not one. You can input the IDs of the categories separate by a comma ','.
6. Deploy your project to a virtual machine on your GCP account. The UI and the API endpoints must be accessible through your public IP address.

Source Code Quality

Two standards will be used to define and measure the quality of your source code.

1. Source Code documentation. You must use the JSDoc library to document your source code for this assignment. (See Assignment Control Panel-->Code Documentation and lab week 4 for help.)
   You are required to provide documentation for your:
   1. classes
   2. global variables and constants
   3. methods
2. Naming Convention. The names of your files, classes, packages, modules, methods and variables must be descriptive and follow the Google naming style. (See Assignment Control Panel-->Code Style for more help.)

Marking Rubric

- No marks will be given to any point you could not explain during the interview
- Not being able to answer multiple questions is a flag for Academic misconduct investigation
- The assignment mark will be set to zero if you miss to submit to Moodle or attend A2 interview
- Source code similarly might be used across all students

Summary

Rubric Breakdown

Expected Output

File Structure

API (New Endpoints)

UI

What to submit?

Where?

Every student must submit assignment-2 to two places: **Moodle** and **Gitlab**.

For Moodle, you must ZIP your files into a single file and upload it to the assignment-2 submission link that can be found on Moodle-->Assessment. There is a quick survey you must do to enable the submission link.

For GitLab, you will be provided with a group repository, and you should commit/push your project to it. We are expecting at least three non-trivial commits to your group repo. See the marking rubric for more details.

⚠ We will mark your assignment based on the latest commit in the `main` branch in your GitLab repository by the due date, **not** in the `master` branch.

What??

1. A document in PDF, DOCX, or MD format that contains:
    1. The routing table for API endpoints requested in Assignment-2 specification.
    2. Steps to run your applications
    3. [Optional] unsolved bugs and issues
    4. this document must be placed in the root folder of your project
2. The source code of the project

Late Submission