CSI 5112 – Software Engineering

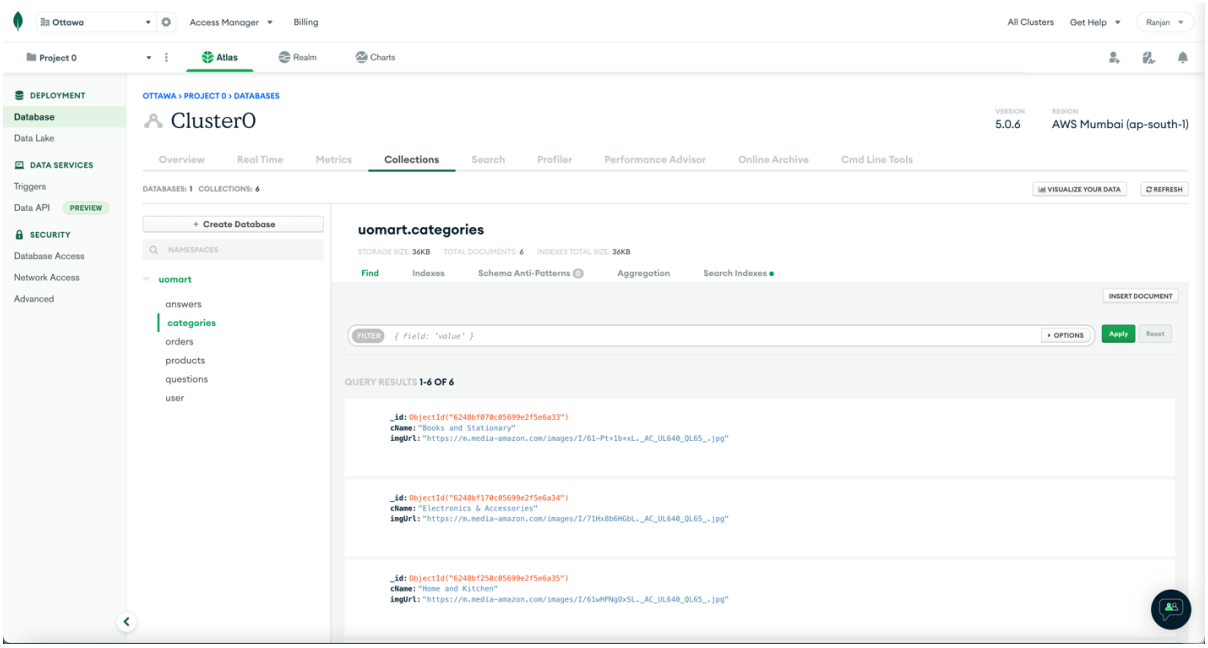
Project - Part 3

Group 12

Functional database integration, CRUD

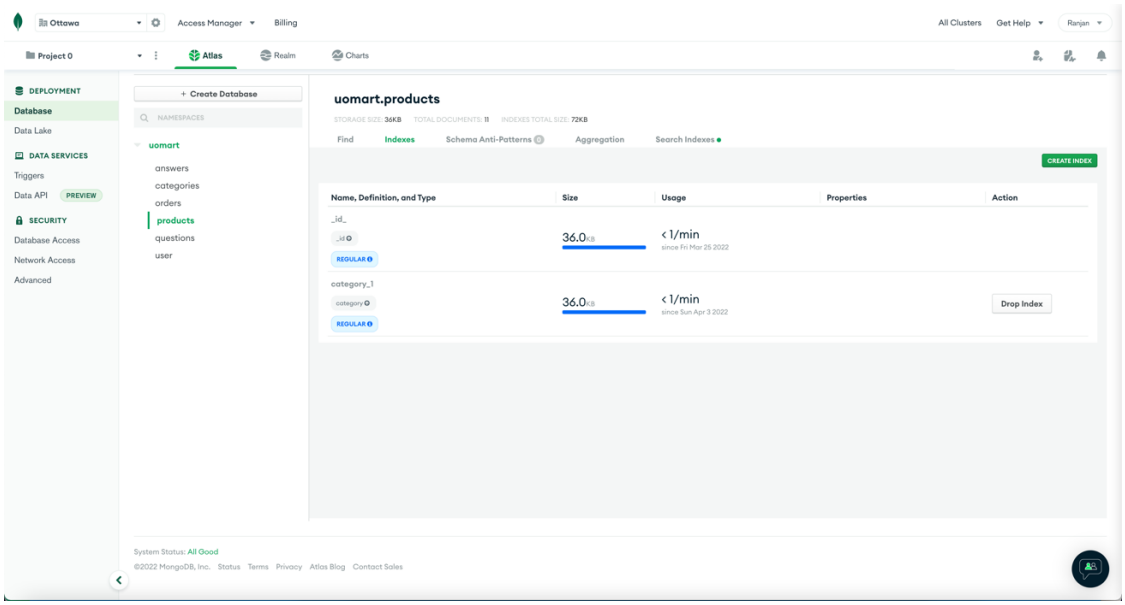
- Which indexes did you create in Mongo? Why?
- Are all CRUD operations working?

Total Collections: 6
Collections: answers, categories, orders, products, questions, user

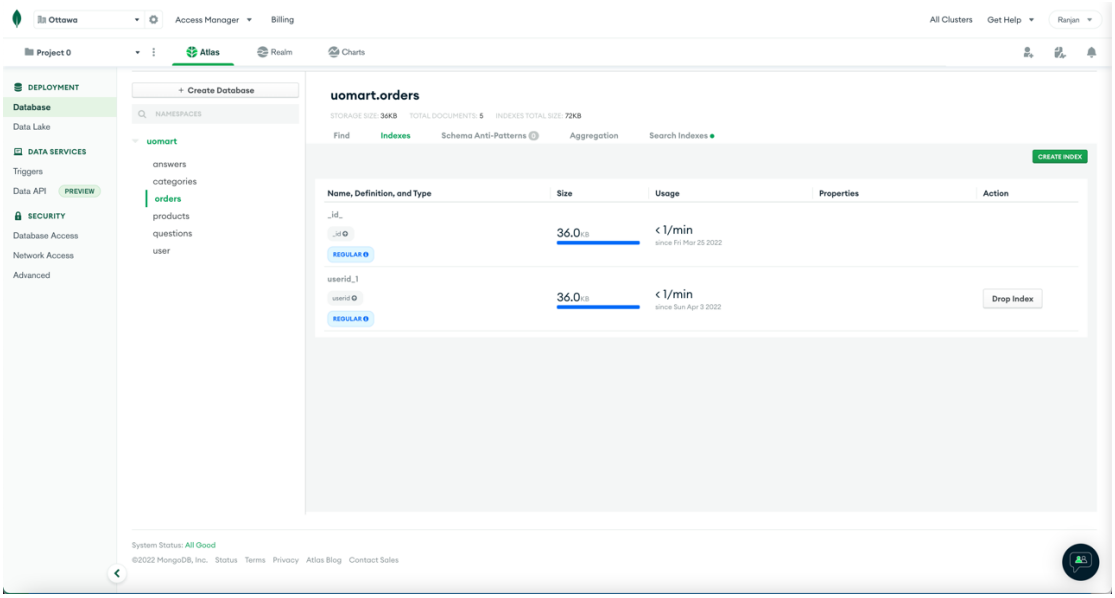


Indexes Created:

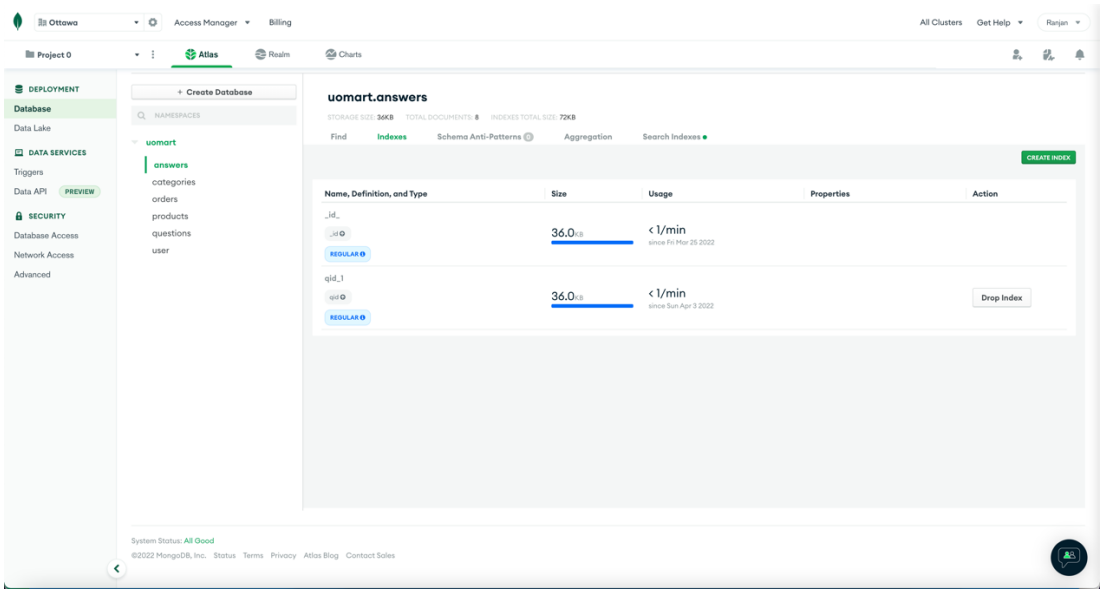
1. **Category Id index on products Collection**
   Since we are fetching products for one category, we created an index for fetching the products using category id.



2. **User Id index on orders Collection**
   We created this as we are fetching the orders for a specific user. Hence, we created an index for fetching the orders using user id.



3. **Question Id index on answers Collection**
   Since we are fetching answers for a particular question, we created an index for fetching the answers using the question id



All the CRUD Operations are working and the API documentation can be accessed from the following link:
https://documenter.getpostman.com/view/11041377/UVyuREpy

Privacy

- Discuss how you would enforce limiting users to only seeing their orders or orders that other users allowed them to see.
- Present a guide of how someone would implement this in your project

Discuss how you would enforce limiting users to only see their orders or orders that other user allowed them to see?

Current Implementation:

In our project, we created an API request in the backend which will only fetch the orders for the user id provided i.e. only the orders for a particular user will be sent as the response from the server. Another approach in our project is that we created a filter in the frontend where the orders will be filtered out for the particular user and the rest will be discarded.

Proposed Solution:

We can use the session store to ensure secure authentication of real data belonging to a particular user, it is beneficial because rather than storing user session data on the application, it's going to store session data within our database. The actual benefit of using a session store is that it will allow access to users based on their role and limit the rights that can be applied at a given time based on their roles defined by the access session. This is one key feature in session management because it refers to a set of requests and responses (viewing of orders) associated

with a certain user and furthermore it supports secure interactions between that user and the application. In addition, session stores are used to keep track of user-specific information, such as database result sets and authorized user IDs, as well as a variety of other activities.

Present a guide of how someone would implement this in your project?

Solution

To develop a session store in MongoDB Database with ASP.NET as Backend, we need to follow the below steps.

1. Firstly, we need to create a TTL Index in our MongoDB Database which will eliminate all the expired session values in a periodic fashion.
2. If suppose, we don't have the ability to generate TTL Index then we can set the option DeleteOldValues to true. Because, if this option is true then in each request to the database, a query will be executed to remove all the expired values.
3. This TTL Index needs to be created in our MongoDB instance.
4. We need to run the following commands in the powershell.

```
dotnet new mvc -au None
dotnet add package Microsoft.AspNetCore.Session --version 2.1.1
dotnet add package MarkCBB.Extensions.Caching.MongoDB --version 2.1.1

dotnet remove package Microsoft.AspNetCore.CookiePolicy

dotnet restore
```

5. In Startup.cs Configuration Services Method, we need to add the following code.

```
services.AddDistributedMongoDBCache(o =>
{
        o.ConnectionString = "<enter a valid connection string>";
        o.DatabaseName = "<enter a valid DatabaseName>";
        o.CollectionName = "<enter a valid CollectionName>";
});

services.AddSession(o =>
{
    o.IdleTimeout = TimeSpan.FromSeconds(60);
    o.Cookie.Name = "Session";
    o.Cookie.HttpOnly = true;
    o.Cookie.SecurePolicy = CookieSecurePolicy.None;
});
```

6. We need to add **app.UseSession()** method before app.UseMvc(routes =>)
7. Now, the session store will be all set.
8. Our final step is to save and execute files which can be done by using below command.

dotnet run

Project report

- Describe challenges and resolutions in each phase of the project. What was the most challenging? What was the most rewarding learning?

Challenges and Resolutions

Phase 1:

Challenge: In the initial stage of the project after gathering all the requirements, the next step was to build the user interface. The first challenge we faced was with designing of the user interface for our application in terms of how an each page should look and what components each page should have.

Resolution: The above issue was unavoidable as the user interface design was one of the most important component in the application design and is directly related to the user experience which may affect the usability of our application and customer satisfaction.

The following are the different strategies we implemented:

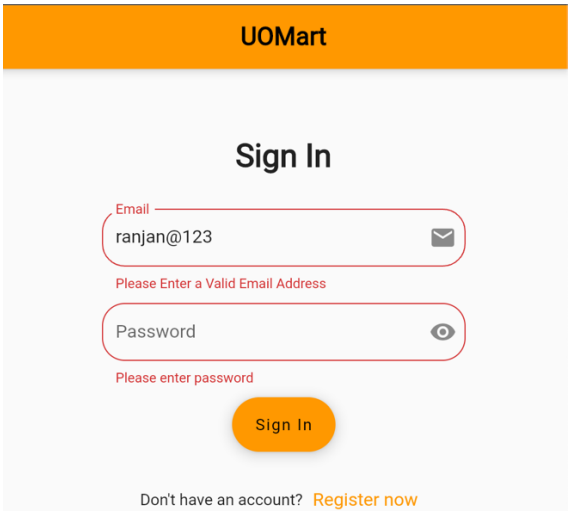
- Referred different articles and blogs to gain knowledge regarding fundamental principles of User Interface Design.
- Examined the interface of popular applications with the similar use case.
- Drawn a rough sketch of each screen using an online Figma. Example of UI Designs made by one of our group members:

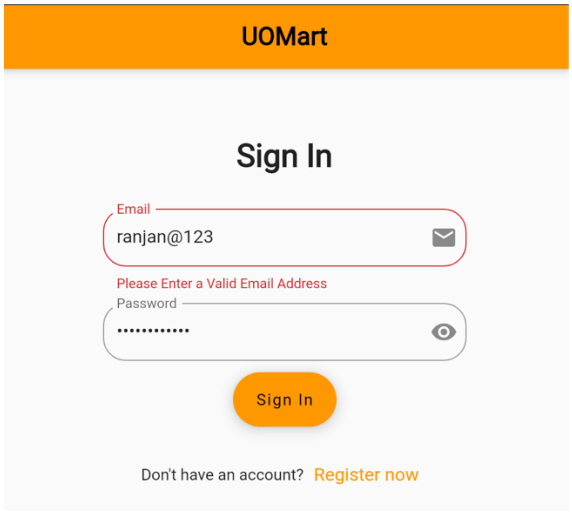
After a continuous refinement we arrived at a final structure and started to build the same using flutter accordingly.

Challenge: Develop Sign In and Signup Screens. This task had some challenges faced by the group members. Firstly, we needed to resolve the issue of size with the input boxes. Secondly, the validations for the input boxes were not working initially. Lastly, the icons in the right side of the inputs, were not working as expected. For example, the show password icon did not perform as expected.

Resolution: Using the SizedBox, we resolved the issue of size with the input boxes. For the validation, we used regular expressions and isEmpty checks and returned appropriate error messages in case the input does not pass the validation checks. Example:

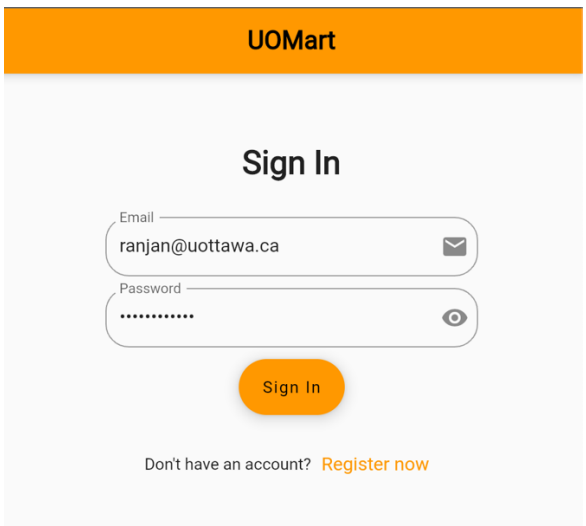
```
SizedBox(
width: 320.0,
child: TextFormField(
  key: const Key("emailAdd"),
  keyboardType: TextInputType.emailAddress,
  onChanged: (value) {
    email = value.toLowerCase();
  },
  decoration: const InputDecoration(
      border: OutlineInputBorder(
        borderSide: BorderSide(width: 2),
        borderRadius: BorderRadius.all(Radius.circular(20.0)),
      ),
      suffixIcon: Icon(Icons.email),
      hintText: 'Enter your email',
      labelText: 'Email'),
  // Validate the input
  validator: (value) {
    if (value!.isEmpty) {
      return 'Please Enter the E-mail Address';
    } else if (!RegExp(
            r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+")
        .hasMatch(email)) {
      return 'Please Enter a Valid Email Address';
    }
    return null;
  },
 ),
),
```

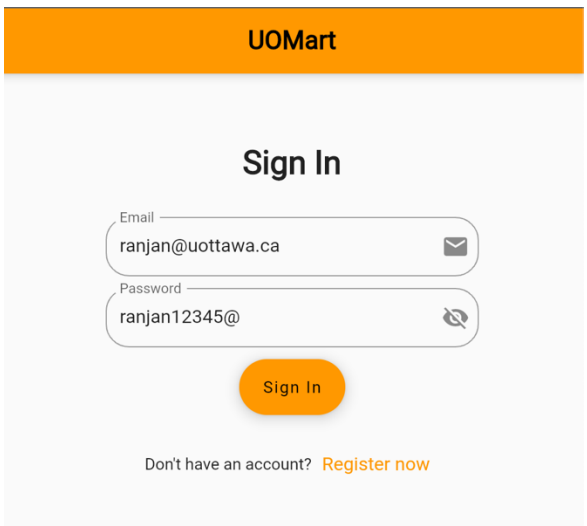




For the icons on the very right side of the input boxes, we used setState in order to render the updates on the screen. For example, in case of password, once the user clicks on show password icon, two things will change, one is the visibility icon and second is the password, which will be then become plain text.

```
suffixIcon: InkWell(
      onTap: () {
        setState(() {
          showPassword = !showPassword;
          visibility == Icons.visibility
              ? visibility = Icons.visibility_off
              : visibility = Icons.visibility;
        });
      },
      child: Icon(visibility),
    ),
```





Challenge: Develop the Cart functionality. For the Customer, we planned to build the cart to be functional in a way that user is able to add or remove items to the cart and the items should stay inside the cart, even when the user navigates through different screens.

Resolution: Using the Provider package [1], we implemented the cart functionality with the necessary functions for adding or removing items from the cart. Similarly, the provider package was also utilized in developing the functionality for user sign in, where the user data is maintained inside the application once the user is signed into the application.

Phase 2:

Challenge: When the backend was implemented, the HTTP GET requests were working properly, but the other requests such as POST, PUT, DELETE were not showing any changes in the backend. we realised that we are using Transient for the service registration.

Resolution: We found out that the transient objects are always different, and a new instance is provided to each service, which means, we will not be able to see any changes in our data [3]. Instead, using singleton, will make sure that the object is same for each request. Therefore, the service registration was changed from transient to singleton. For example, we changed the following code:

builder.Services.AddTransient<OrderService>();

to:

builder.Services.AddSingleton<OrderService>();

Challenge: When the category is modified, or say new answer is posted for the question in the discussion, the screen did not update. For example, let's say that we tried to update one of the categories, "Clothing and Shoes" to "Clothing and Footwear". The moment we sent the request, the category did update, but not on the screen, therefore, we will not see changes, unless we go back and push the screen again.

Resolution: To resolve this issue, we used then() method after Navigator.push [2], where we fetched the categories again inside the setState method, and at places, where we are updating the changes on the same screen, we simply called setState method. Example, for adding a new category:

```
Navigator.of(context).pushNamed('/add-category').then((value) {
        setState(() {
          futureCategories = fetchCategories();
        });
      });
```

This will now update the list on the screen the moment we create a new category and come back to the categories screen.

Phase 3:

Challenge: While connecting MongoDB with the backend, we realized that previously, we were creating and using our own ids whereas, now with the MongoDB, the ids are automatically getting generated. The issue was to solve the inconsistency of ids with backend and frontend.

Resolution: In order to maintain consistency between frontend and backend with the MongoDB, we kept the variables name intact but while sending and retrieving the ids from MongoDB. Example, in category, *cid* is mapped to the id obtained from MongoDB:

```
public class Category
{
    [BsonId][BsonRepresentation(BsonType.ObjectId)]
    public string? cid { get; set; }
    [BsonElement]
    public string cName { get; set; }
    [BsonElement]
    public string imgUrl { get; set; }

    public Category(string Cid, string CName, string ImgUrl)
    {
        this.cid = Cid;
        this.cName = CName;
        this.imgUrl = ImgUrl;
    }
}
```

The most challenging part was the functionality to update the categories on the screen after adding or modifying the details of a category and returning back to the categories screen. This was most challenging as it was difficult to design the exact structure required to implemented in order for this to work seamlessly as we are updating the details from the backend.

The most rewarding learning was the use of provider package in flutter. It is a very powerful way of state management by “lifting the state up”, which means that we keep the state above the widgets that use it. For our project, we kept the providers at root level in order to use it for all the widgets in the application.

REFERENCES:

[1] Provider flutter package - https://pub.dev/packages/provider

[2] https://stackoverflow.com/questions/49804891/force-flutter-navigator-to-reload-state-when-popping

[3] Transient and Singleton - https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences