Ilyass Hmamou
ilyass.hmamou@snhu.edu

CS-499: Computer Science Capstone
7-1: Final project

Table of contents

Southern New Hampshire University

1- Artifact description:

Throughout this journey, I chose to focus on enhancing the "Salvare Search For Rescue App" artifact. As a quick reminder, this artifact was created between September 2023 and October 2023 as part of the final project for the CS-340: Client/Server development course.

The app allows users to interact with data from an existing animal shelter database through two distinct methods: an interactive datatable with built-in filters and an interactive map presented in a user-friendly interface. Developed using the Python language, pyMongo driver, Dash framework, and MongoDB, the application delivers a holistic user experience.

2- Enhancements Summary:

The "Salvare Search For Rescue App" offers significant enhancement possibilities that fulfill all the requirements for this course.

In the realm of software design and engineering, the artifact underwent improvements by implementing local MongoDB setup to run the application. The backend, front-end, and database code were enhanced to ensure security, efficiency, scalability, and maintainability—essential components in the software design and engineering category.

Within the Algorithm and Data Structure category, the focus was on elevating code quality. This involved transforming a nested if-else code block, considered a suboptimal algorithm and code design practice, into a switch-case-like design. This improvement was

implemented by creating functions and utilizing map dictionaries data structure, meeting the criteria for introducing a new data structure in this enhancement.

In the databases category, the emphasis shifted to the DevOps aspect of database experience. This was demonstrated by showcasing skills in hosting the dataset on the cloud. The process involved researching cloud solutions, setting up the cloud server, creating cloud users, importing the dataset, and ultimately integrating with a third-party GUI. This integration resulted in displaying data to end-users through a user-friendly UI that adheres to all UI/UX requirements.

3- Software design and Engineering enhancement

3-1 enahncement details:

The enhancement of this artifact involved two key steps. Firstly, the project was replicated locally, providing full control over the development environment. This not only facilitates the current enhancement but also establishes a foundation for future improvements, especially with regard to an upcoming database enhancement. This step showcases proficiency in the DevOps domain, involving the creation of a Python virtual environment, configuration of MongoDB, importation of necessary libraries, and integration of the database.

The second step focuses on code updates, encompassing modifications to both the front-end and back-end. Front-end enhancements aim to improve user-friendliness and introduce additional functionalities, such as buttons for filtering by animal type—an absent feature in the

original build. On the back-end, a server-side filtering mechanism was implemented to allow users to filter data by animal type directly from the backend, bypassing client-side filtering within the browser UI. This update is motivated by the goal of enhancing application performance, recognizing that app servers typically possess more computational power than client machines where client-side filtering usually occurs. The following section provides more detailed insights into the implementation of each of these two steps.

3-2- Replicated the project locally:

- I set up a Python virtual environment for Jupyter on my local machine (following steps documented here:https://sesync-ci.github.io/faq/python-virtual-env.html)
- Install Mongo db locally from the Mongodb official website link: https://www.mongodb.com/docs/manual/installation/
- Set up local user for mongodb
- Imported the database using the mongoimport command

3-3: code updates:

Front-end updates:

- In the dashboard layout section, I updated the html div responsible on adding the Grazioso Salvare Logo image to the site, to resize the image to not take entire half of the screen and move it to the side.

  Below screenshot of the initial code Vs the new code

  Initial:

```
app.layout = html.Div([
        html.Center(html.Img(src='{}'.format(encoded_image.decode()))),
    html.Center(html.B(html.H1('[Ilyass Hmamou] - CS-340 Dashboard'))),
```

New:

```
app.layout = html.Div([
    #html.Center(html.Img(src='{}'.format(encoded_image.decode()))),
    html.A([
            html.Img(
                src='{}'.format(encoded_image.decode()),
                style={
                    'height' : '25%',
                    'width' : '25%',
                    'float' : 'right',
                    'position' : 'relative',
                    'padding-top' : 0,
                    'padding-right' : 0
                })
    ], href='https://www.snhu.edu'),
    html.Center(html.B(html.H1('[Ilyass Hmamou] - CS-340 Dashboard'))),
```

- In the same section (dashboard layout) updated the part responsible on creating the radio buttons for the Rescue Type Filter to be a dropdown menu, and added buttons to search by animal_type (as in the App screenshots in the **Before and After** section)

Initial code:

```
html.Hr(),
html.Div(
    #Radio Items to select the rescue filter options
    dcc.RadioItems(
        id='filter-type',
        # created the labels and keys based on the Grazioso requirements
        options=[
        {'label': 'Water Rescue', 'value': 'WR'},
        {'label': 'Mountain/Wilderness Rescue', 'value': 'MWR'},
        {'label': 'Disaster Rescue/Individual Tracking', 'value': 'DRIT'},
        {'label': 'Show All', 'value': 'All'}
        ],
        value='All',
        labelStyle={'display': 'inline-block'}
    )
),
html.Hr(),
dt.DataTable(
    id='datatable-id',
    columns=[
        {"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns
    ],
    data=df.to_dict('records'),
```

New:

```
html.Hr(),
# buttons at top of table to filter the data set to find cats or dogs
html.Div(className='row',
    style={'display' : 'flex'},
    children=[
        html.Span("Filter by:", style={'margin': 6}),
        html.Span(
            html.Button(id='submit-button-one', n_clicks=0, children='Cats'),
            style={'margin': 6}
        ),
        html.Span(
            html.Button(id='submit-button-two', n_clicks=0, children='Dogs'),
            style={'margin': 6}
        ),
        html.Span(
            html.Button(id='reset-buttons', n_clicks=0, children='Reset', style={'background-color': 'red', '
            style={'margin': 6,}
        ),
```

```
        ,,
        #dropdown for Rescue type
        html.Span("or", style={'margin': 6}),
        html.Span([
            dcc.Dropdown(
                id='filter-type',
                options=[
                    {'label': 'Water Rescue', 'value': 'wr'},
                    {'label': 'Mountain or Wilderness Rescue', 'value': 'mwr'},
                    {'label': 'Disaster Rescue or Individual Tracking', 'value': 'drit'}
                ],
                placeholder="Select a Dog Rescue Category Filter",
                style={'marginLeft': 5, 'width': 350}
            )
        ])
    ]
),
html.Hr(),
dt.DataTable(
    id='datatable-id',
    columns=[
        {"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns
    ],
    data=df.to_dict('records'),
```

Back-end updates:

- Updated the app callback action for the dashboard update method to get triggered by the created dropdown and buttons instead of the radio button that the app was initially using.

  Initial:

```
#############################################
# Interaction Between Components / Controller
#############################################

@app.callback([Output('datatable-id','data'),
               Output('datatable-id','columns')],
              [Input('filter-type', 'value')])
def update_dashboard(filter_type):
        #filter interactive data table with MongoDB queries
        if filter_type == 'WR':
```

New:

```
# This callback add interactive dropdown filter option to the dashboard to find dogs per category
# or interactive button filter option to the dashboard to find all cats or all dogs
@app.callback(
    Output('datatable-id', 'data'),
    [Input('filter-type', 'value'),
     Input('submit-button-one', 'n_clicks'),
     Input('submit-button-two', 'n_clicks')]
)
def update_dshboard(selected_filter, btn1, btn2):
```

- Updated the method update_dashboard responsible of getting the data from database by calling the CRUD methods implemented in the AnimalShelter class.

Initial:

```
def update_dashboard(filter_type):
        #filter interactive data table with MongoDB queries
        if filter_type == 'WR':
            df = pd.DataFrame(list(db.readAll({
                "breed" : {"$in": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]},
                "sex_upon_outcome" : "Intact Female",
                "age_upon_outcome_in_weeks": {"$lte": 156, "$gte": 26}})))
        elif filter_type == 'MWR':
            df = pd.DataFrame(list(db.readAll({
                "breed" : {"$in": ["German Shephard", "Alaskan Malamute", "Old English Sheepdog",
                                   "Siberian Husky", "Rottweiler"]},
                "sex_upon_outcome" : "Intact Male",
                "age_upon_outcome_in_weeks": {"$lte": 156, "$gte": 26}})))
        elif filter_type == 'DRIT':
            df = pd.DataFrame(list(db.readAll({
                "breed" : {"$in": ["Doberman Pinscher", "German Shephard", "Golden Retriever",
                                   "Bloodhound", "Rottweiler"]},
                "sex_upon_outcome" : "Intact Male",
                "age_upon_outcome_in_weeks": {"$lte": 300, "$gte": 20}})))
        elif filter_type == 'All':
            df = pd.DataFrame.from_records(db.readAll({}))


        columns=[{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns]
        data=df.to_dict('records')

        return (data,columns)
```

New:

Code for the dropdown filter:

```
def update_dshboard(selected_filter, btn1, btn2):
    if (selected_filter == 'drit'):
        df = pd.DataFrame(list(db.readAll(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":[
                        "Doberman Pinscher","German Shepherd","Golden Retriever","Bloodhound","Rottweiler"]},
                    "sex_upon_outcome":"Intact Male",
                    "age_upon_outcome_in_weeks": {"$gte":20},
                    "age_upon_outcome_in_weeks":{"$lte":300}
                }
            )
        ))
    elif (selected_filter == 'mwr'):
        df = pd.DataFrame(list(db.readAll(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":[
                        "German Shepherd","Alaskan Malamute","Old English Sheepdog",
                        "Siberian Husky","Rottweiler"]},
                    "sex_upon_outcome":"Intact Male",
                    "age_upon_outcome_in_weeks":{"$gte":26},
                    "age_upon_outcome_in_weeks":{"$lte":156}
                }
            )
        ))
```

```
    elif (selected_filter == 'wr'):
        df = pd.DataFrame(list(db.readAll(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":["Labrador Retriever Mix","Chesapeake Bay Retriever","Newfoundland"]},
                    "sex_upon_outcome":"Intact Female",
                    "age_upon_outcome_in_weeks":{"$gte":26},
                    "age_upon_outcome_in_weeks":{"$lte":156}
                }
            )
        ))
```

Code for the buttons

```
    # higher number of button clicks to determine filter type
    elif (int(btn1) > int(btn2)):
        df = pd.DataFrame(list(db.readAll({"animal_type":"Cat"})))
    elif (int(btn2) > int(btn1)):
        df = pd.DataFrame(list(db.readAll({"animal_type":"Dog"})))
    else:
        df = pd.DataFrame.from_records(db.readAll({}))

    data = df.to_dict('records')

    return data
```

3-4: Course objective and skills showcased

This Artifact and the enhancement made to it, showcases the outcome related to

demonstrating an ability to use well-founded and innovative techniques, skills, and tools in

computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.

Upon completing the implementation, it is apparent that this goal was successfully realized. This accomplishment was made possible by integrating class design techniques and leveraging expertise in both back-end and front-end languages to develop the application and implement significant updates for the end users.

The skills utilized to attain this result encompass the following:

1. Establishing a local environment to replicate the application locally, demonstrating proficiency in the DevOps domain, a crucial aspect for initiating any programming project.

2. Performing code updates across the back-end, front-end, and database, showcasing coding skills indicative of a full-stack developer.

3. Introducing the AnimalShelter class in a distinct file, separated from the utility file "projectTwoDash," illustrating a comprehension of Object-Oriented Programming concepts.

4. Evaluating the existing application, identifying weaknesses, and implementing effective solutions, highlighting analytical skills and innovative techniques as a programmer.

4- Algorithm and datastructure enhancement

4-1 Enhancement details

The focus on this enhancement was to improve the Algorithm used on the server side filtering logic, which will lead to improve the efficiency of the code. This code improvement plan was around eliminating a nested if-elif block in the update_dashboard() method and replace

it with a switch case like logic; this change required the usage of dictionary data structure since Python doesn't have a built-in switch case statement, therefore, I used a dictionary to map cases to functions, and provide a similar case of switch case. On this report, I will provide a deep dive on the technical details for this enhancement.

4-2 code updates

v1: original artifact code:

```
@app.callback([Output('datatable-id','data'),
               Output('datatable-id','columns')],
              [Input('filter-type', 'value')])
def update_dashboard(filter_type):
        #filter interactive data table with MongoDB queries
        if filter_type == 'WR':
            df = pd.DataFrame(list(db.readAll({
                "breed" : {"$in": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]},
                "sex_upon_outcome" : "Intact Female",
                "age_upon_outcome_in_weeks": {"$lte": 156, "$gte": 26}})))
        elif filter_type == 'MWR':
            df = pd.DataFrame(list(db.readAll({
                "breed" : {"$in": ["German Shephard", "Alaskan Malamute", "Old English Sheepdog",
                                   "Siberian Husky", "Rottweiler"]},
                "sex_upon_outcome" : "Intact Male",
                "age_upon_outcome_in_weeks": {"$lte": 156, "$gte": 26}})))
        elif filter_type == 'DRIT':
            df = pd.DataFrame(list(db.readAll({
                "breed" : {"$in": ["Doberman Pinscher", "German Shephard", "Golden Retriever",
                                   "Bloodhound", "Rottweiler"]},
                "sex_upon_outcome" : "Intact Male",
                "age_upon_outcome_in_weeks": {"$lte": 300, "$gte": 20}})))
        elif filter_type == 'All':
            df = pd.DataFrame.from_records(db.readAll({}))


        columns=[{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns]
        data=df.to_dict('records')

        return (data,columns)
```

Updating this if elif block to switch case

V2: post enhancement one artifact code:

```
def update_dshboard(selected_filter, btn1, btn2):
    #if Disaster Rescue or Individual Tracking option selected
    if (selected_filter == 'drit'):
        # call the readAll method from the AnimalShelter class
        # pass parameters identifying the selected type of rescue
        df = pd.DataFrame(list(db.readAll(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":[
                        "Doberman Pinscher","German Shepherd","Golden Retriever","Bloodhound","Rottweiler"]},
                    "sex_upon_outcome":"Intact Male",
                    "age_upon_outcome_in_weeks": {"$gte":20},
                    "age_upon_outcome_in_weeks":{"$lte":300}
                }
            )
        ))
    # if Mountain or Wilderness Rescue option is selected
    elif (selected_filter == 'mwr'):
        # call the readAll method from the AnimalShelter class
        # pass parameters identifying the selected type of rescue
        df = pd.DataFrame(list(db.readAll(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":[
                        "German Shepherd","Alaskan Malamute","Old English Sheepdog",
                        "Siberian Husky","Rottweiler"]},
                    "sex_upon_outcome":"Intact Male",
                    "age_upon_outcome_in_weeks":{"$gte":26},
                    "age_upon_outcome_in_weeks":{"$lte":156}
                }
            )
```

Updating if elif block to switch case

```
            )
        ))
    # if Water Rescue option is selected
    elif (selected_filter == 'wr'):
        # call the readAll method from the AnimalShelter class
        # pass parameters identifying the selected type of rescue
        df = pd.DataFrame(list(db.readAll(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":["Labrador Retriever Mix","Chesapeake Bay Retriever","Newfoundland"]},
                    "sex_upon_outcome":"Intact Female",
                    "age_upon_outcome_in_weeks":{"$gte":26},
                    "age_upon_outcome_in_weeks":{"$lte":156}
                }
            )
        ))
    # higher number of button clicks to determine filter type
    elif (int(btn1) > int(btn2)):
        # call the readAll method from the AnimalShelter class
        # pass parameters animal_type = Cat
        df = pd.DataFrame(list(db.readAll({"animal_type":"Cat"})))
    elif (int(btn2) > int(btn1)):
        # call the readAll method from the AnimalShelter class
        # pass parameters animal_type = Dog
        df = pd.DataFrame(list(db.readAll({"animal_type":"Dog"})))
    else:
        # when reset button is clicked, call readAll method with no parameters to return all the data
        df = pd.DataFrame.from_records(db.readAll({}))

    data = df.to_dict('records')
```

updating if elif block to switch case

updated code:

Let's walk through the refactored code step by step:

First, we define individual functions for each filter type. These functions return the filter criteria based on the selected filter type.

```
disaster_rescue_filter():
return {
    "animal_type": "Dog",
    "breed": {"$in": ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"]},
    "sex_upon_outcome": "Intact Male",
    "age_upon_outcome_in_weeks": {"$gte": 20, "$lte": 300}
}
```

```
mountain_wilderness_rescue_filter():
return {
    "animal_type": "Dog",
    "breed": {"$in": ["German Shepherd", "Alaskan Malamute", "Old English Sheepdog", "Siberian Husky", "Rottweil
    "sex_upon_outcome": "Intact Male",
    "age_upon_outcome_in_weeks": {"$gte": 26, "$lte": 156}
}
```

```
water_rescue_filter():
return {
    "animal_type": "Dog",
    "breed": {"$in": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]},
    "sex_upon_outcome": "Intact Female",
    "age_upon_outcome_in_weeks": {"$gte": 26, "$lte": 156}
}
```

```
cat_filter():
return {"animal_type": "Cat"}
```

```
dog_filter():
return {"animal_type": "Dog"}
```

```
reset_filter():
return {}
```

Next, we create a dictionary (`filter_cases`) to map filter types to their corresponding functions. Then, refactor the main function that updates the dashboard (`update_dashboard`) to use the dictionary to get the appropriate filter function based on the selected filter type.

```
update_dashboard(selected_filter, btn1, btn2):
filter_cases = {
    'drit': disaster_rescue_filter,
    'mwr': mountain_wilderness_rescue_filter,
    'wr': water_rescue_filter,
    'cat': cat_filter,
    'dog': dog_filter,
    'reset': reset_filter
}

selected_filter_fn = filter_cases.get(selected_filter, reset_filter)
df = pd.DataFrame(list(db.readAll(selected_filter_fn())))

data = df.to_dict('records')
return data
```

explanation:

- `selected_filter` is the user-selected filter type.

- `filter_cases.get(selected_filter, reset_filter)` retrieves the corresponding filter function from the dictionary. If the selected filter is not found, it defaults to the `reset_filter` function.

- The function then fetches data from the database using the selected filter criteria and converts it to a format suitable for the dashboard.

The function that resets the clicks of the cat and dog filter buttons (`update_buttons`) has been modified to use `PreventUpdate` when the reset button is clicked.

```
def update_buttons(reset):
    if reset:
        return 0, 0
    else:
        raise PreventUpdate
```

Explanation:

- If the reset button is clicked (`reset` is truthy), it returns 0 for both buttons to reset their click counts.

- If the reset button is not clicked, it raises `PreventUpdate` to prevent undesired updates.

Finally, the Dash callbacks are simplified to directly call the respective functions.

```
# This callback add interactive dropdown filter option to the dashboard to find dogs per category
# or interactive button filter option to the dashboard to find all cats or all dogs
@app.callback(
    Output('datatable-id', 'data'),
    [Input('filter-type', 'value'),
     Input('submit-button-one', 'n_clicks'),
     Input('submit-button-two', 'n_clicks')]
)
def callback_update_dashboard(selected_filter, btn1, btn2):
    return update_dashboard(selected_filter, btn1, btn2)
```

```
# This callback reset the clicks of the cat and dog filter button
@app.callback(
    [Output('submit-button-one', 'n_clicks'),
     Output('submit-button-two', 'n_clicks')],
    [Input('reset-buttons', 'n_clicks')]
)
def callback_update_buttons(reset):
    return update_buttons(reset)
```

These callbacks directly call the `update_dashboard` and `update_buttons` functions, improving readability and maintaining the separation of concerns.

In summary, the refactoring uses functions and a dictionary to organize and streamline the code, making it more modular, readable, and maintainable. Each function has a specific responsibility, and the dictionary provides an efficient way to handle multiple filter cases.

4-3 Benifits of the updated code

The refactored code using the dictionary approach is not necessarily more efficient in terms of computational performance. The primary advantage lies in improved readability, maintainability, and modularity. Let me explain:

1. Readability:

- The use of a dictionary to map cases to functions makes it clear which function corresponds to each case. This can enhance the readability of your code, especially when there are multiple cases.

2.Maintainability:

- If you need to add or modify cases, you can do so by simply updating the `filter_cases` dictionary. This makes the code easier to maintain and extend. In the original `if-elif-else` block, adding or modifying cases might involve changing multiple locations in the code, making it more error-prone.

3. Modularity:

- Each filter is encapsulated within its own function, promoting a more modular design. This makes it easier to understand each individual case and modify behavior without affecting the rest of the code.

4. Reduced Code Duplication:

- The dictionary approach eliminates the need to repeat the call to `pd.DataFrame(list(db.readAll(...)))` for each case, reducing code duplication.

5. Flexibility:

- The dictionary approach allows for the inclusion of more complex logic within each filter function if needed. This can be beneficial as the codebase evolves.

It's important to note that these advantages come from a design and maintenance perspective. In terms of runtime efficiency, the two approaches are likely to be similar, and the choice between them often depends on factors like codebase size, team preferences, and future extensibility requirements. If performance is a critical concern, the focus should be on optimizing the logic within the filter functions rather than the structure of the control flow.

6. Time complexity:

In terms of time complexity, both approaches are reasonable, and the difference is often negligible. The primary advantage of using a dictionary is improved readability and maintainability, especially when dealing with a large number of conditions. The dictionary allows for direct mapping of cases to functions without the need for sequential evaluation.

Therefore, the switch case model using a dictionary is preferred in Python for its readability and maintainability, even if the time complexity is similar to the original `if-elif` structure. The

benefits of code organization and ease of understanding often outweigh minor differences in time complexity for these types of structures.

7. Optimization:

The switch case using a dictionary is generally favored in Python for its readability and maintainability advantages. While both approaches have similar time complexities, the switch case using a dictionary allows for a cleaner organization of code, easier modification, and reduced redundancy.

In practice, the switch case using a dictionary is often considered a more "Pythonic" and idiomatic approach when dealing with multiple conditions. However, the best approach depends on the specific context and requirements of your application.

8. Efficiency:

In terms of efficiency, the differences between the switch case using a dictionary and the original `if-elif` structure are likely to be minimal in most practical scenarios. The primary considerations should be code readability, maintainability, and ease of modification.

The switch case using a dictionary is often favored in Python for its clean and idiomatic design. The efficiency gains, if any, are likely to be marginal compared to the benefits of improved code organization and readability. Ultimately, the choice between the two approaches should consider the specific requirements and preferences of your project.

4-4: Course objective and skills showcased

This enhancement showcases two course outcomes:

1- 'Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices'

2- 'Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts'

These results were attained by analyzing the existing code and pinpointing algorithmic shortcomings, such as the utilization of nested if-elif clauses, the code exhibiting a waterfall structure, and a deficiency in coding efficiency resulting in poor readability and maintainability. These issues address the first outcome of devising and assessing computational solutions that resolve a given problem through algorithmic principles. Subsequently, these concerns were rectified by introducing a more efficient code implementation and furnishing a comprehensively documented explanation, as illustrated in the step-by-step walkthrough outlined above. This addresses the secondary outcome.

The skills utilized to attain this result encompass the following:

1. Evaluating the current code and pinpointing areas of concern, such as scalability, maintainability, and readability.

2. Devising an algorithmic solution that seamlessly translates into code to address the identified issues.

3. Recognizing the significance of code efficiency and the need for comprehensive documentation.

4. Proficiency in generating technical documentation outlining the necessary modifications, coupled with a step-by-step elucidation of the proposed solution.

5- Databases enhancement

5-1 Enhancement details:

As previously stated, this improvement centered on transferring the data to a cloud server and establishing a straightforward method to link to the server and engage with the data through a third-party integrated UI tool. In this segment of the documentation, I will outline a step-by-step configuration process pertaining to configuring the database on the server, generating user profiles for database access, and finally establishing a connection to the database using the third-party UI tool.

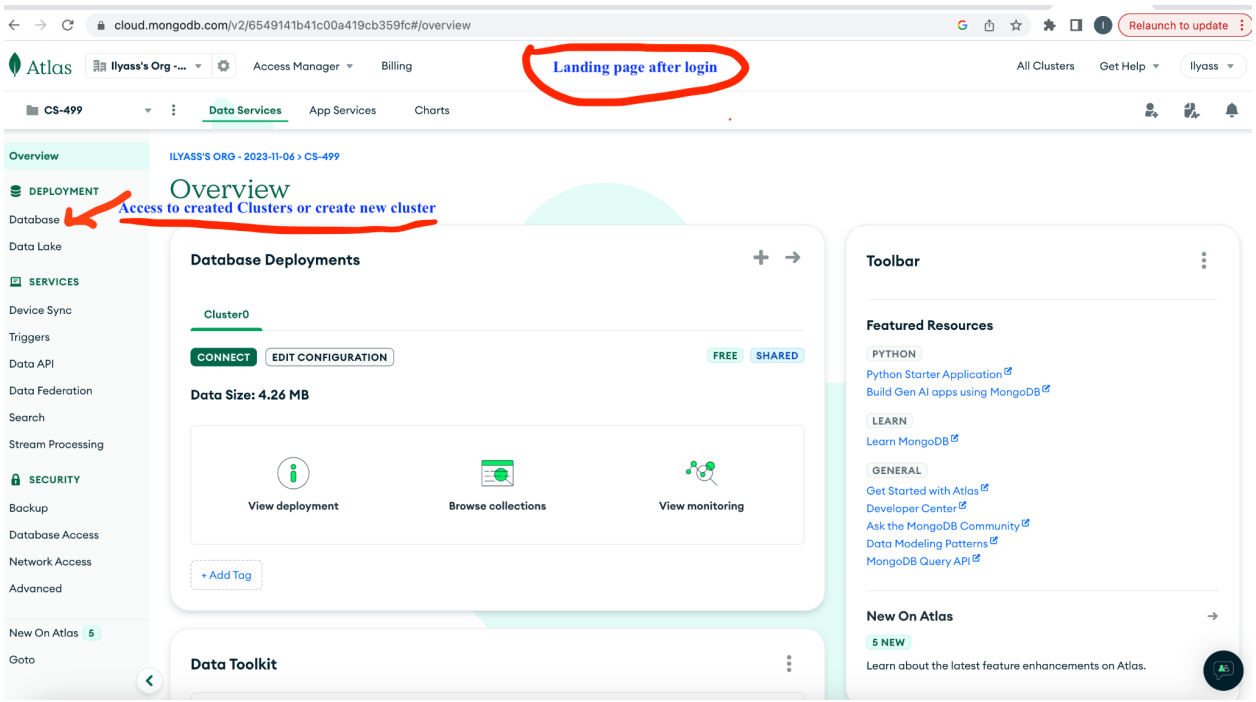
5-2 Cloud database details:

The server used is MongoDB Atlas which is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS , Azure, and GCP).

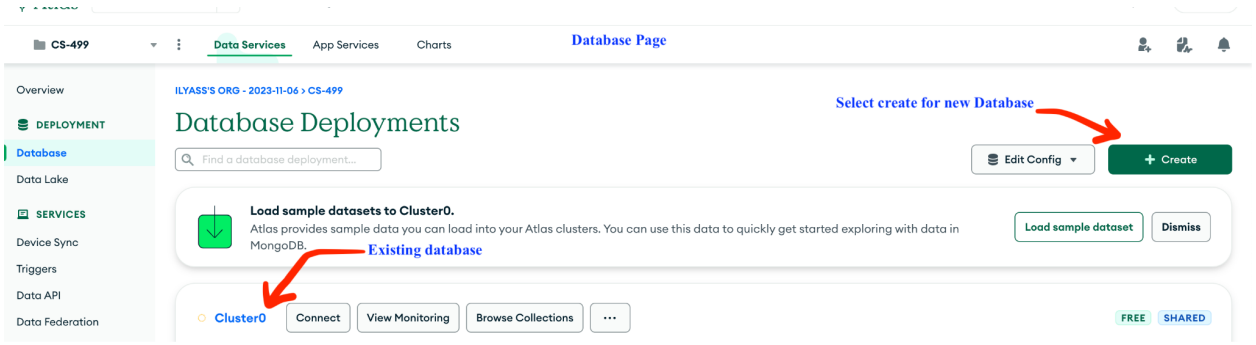
Link: https://www.mongodb.com/

5-3 Database creation within Atlas:

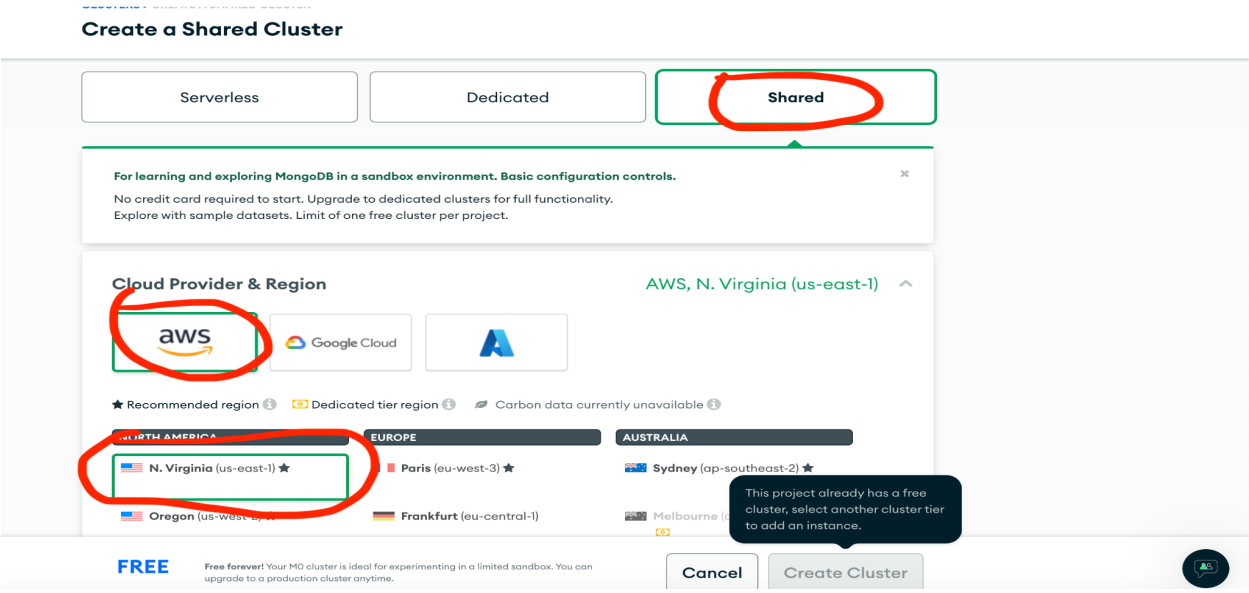
Once you login to Mongodb Atlas, you will land on the following page:



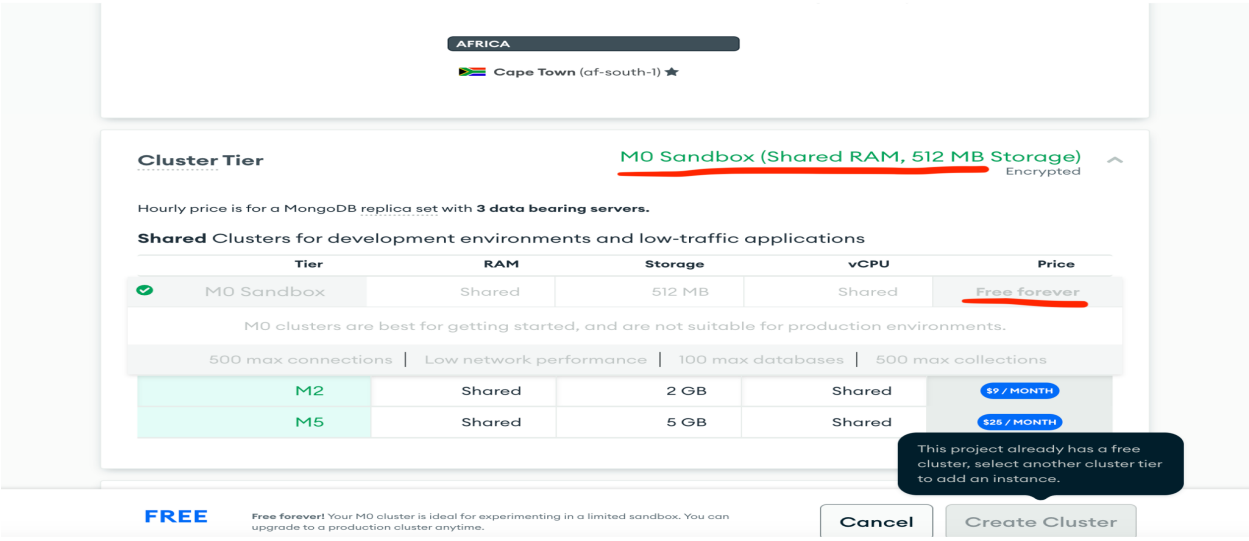
Click on the Database option on the right side menu



Once you select Create, you get the following page where you can choose your host cloud provider and your preferred location/timezone



For this course, I'm using the Shared Cluster (has a free trial option), with AWS as the cloud provider and N.Virginia as the region.

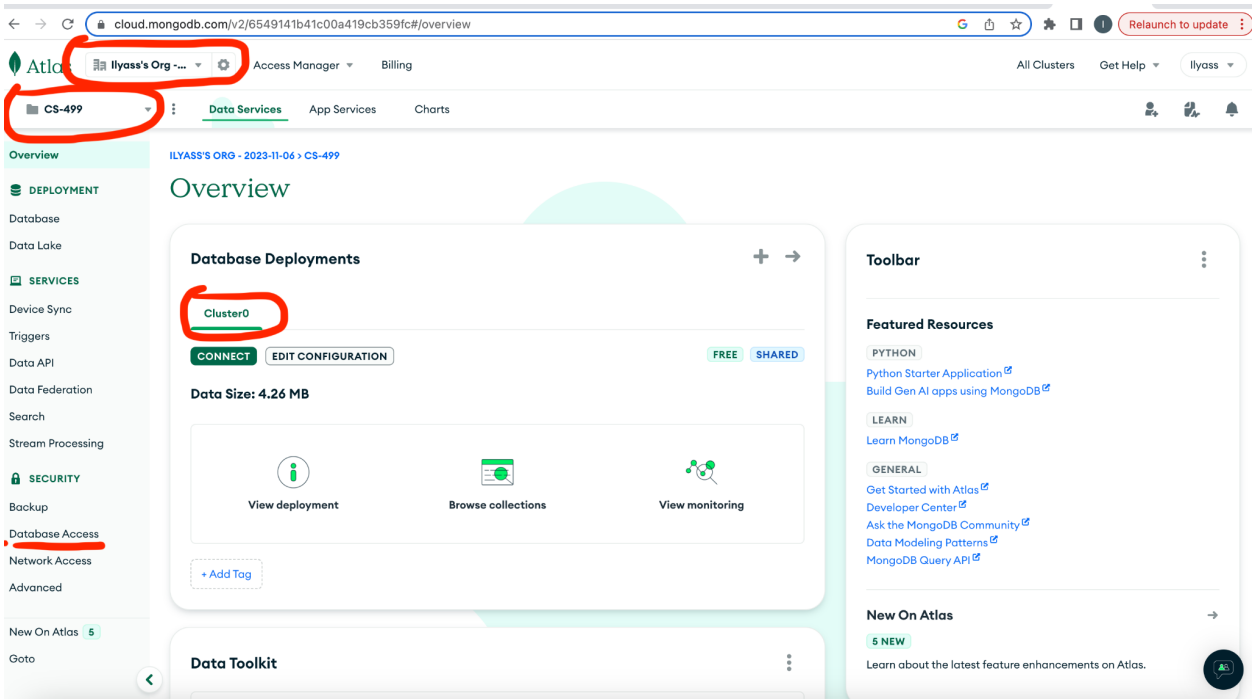


Once you select the cluster Tier you want to have, select Create Cluster.

Note: The host permits only one complimentary sandbox cluster per account. The "Create Cluster" option is deactivated in this screenshot since I had already established a cluster a few weeks ago for the purpose of this course.

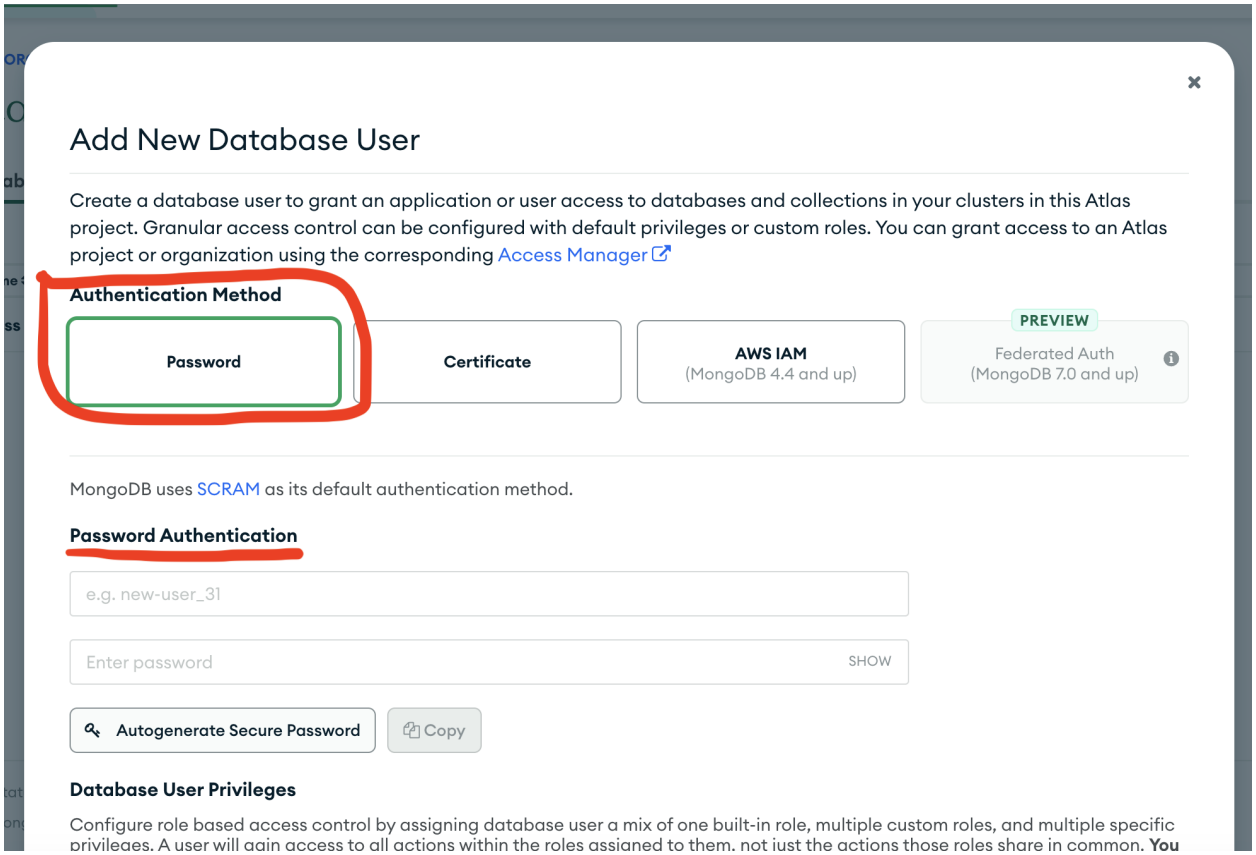
5-4 New user creation

After creating the database cluster, the next step is to configure a user. Ensure that you are on the landing page within the desired project and cluster. Then, choose "Database access" from the menu on the right side.



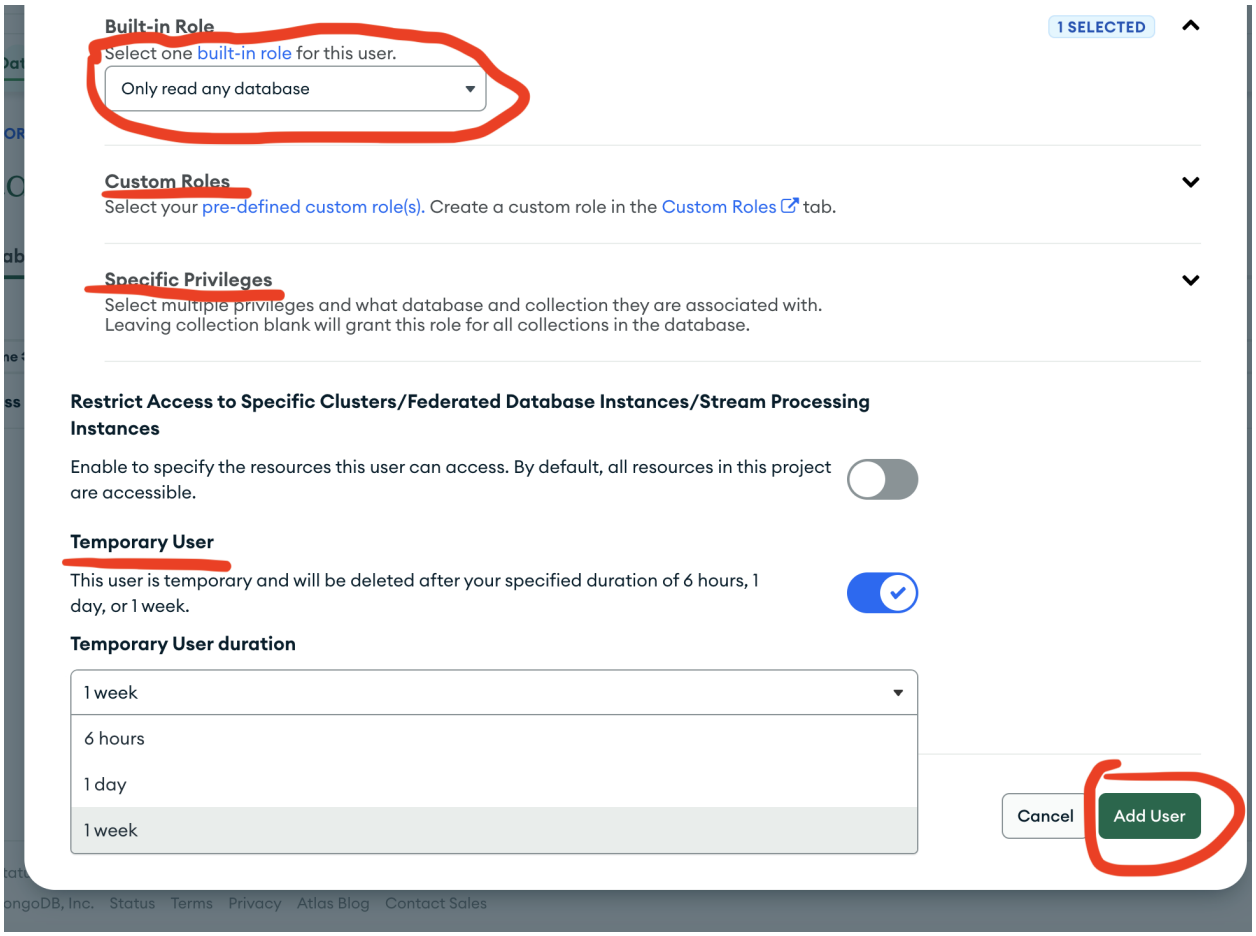
On the next page, select “Add new Database user”

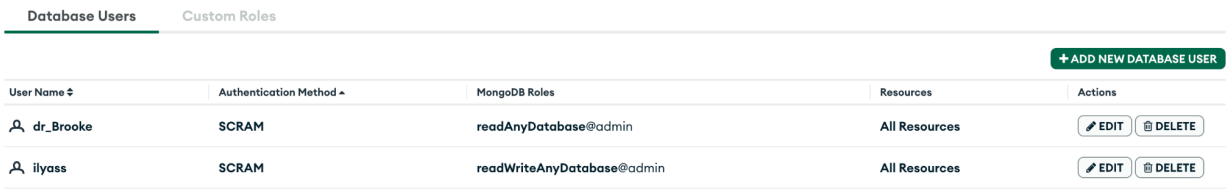
Next, pick your authentication method. In this course, I opted for the Username and Password authentication method.



In the upcoming section, configure the user's permissions. You have the option to select Built-in permissions, providing choices like read-only, read and write, or admin. Alternatively, you can

opt for a more advanced option offering detailed permission settings. For the simplicity of this course, I've chosen one of the Built-in permissions to streamline the process.

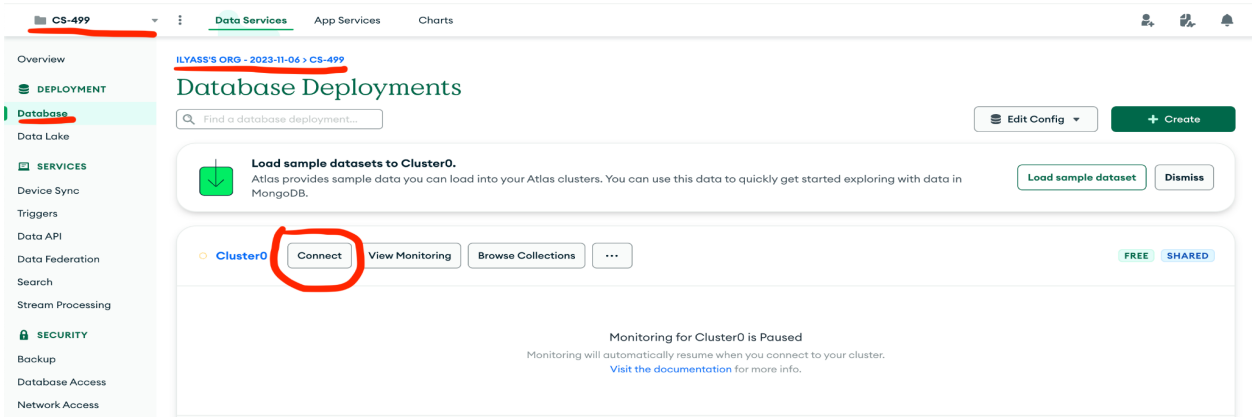




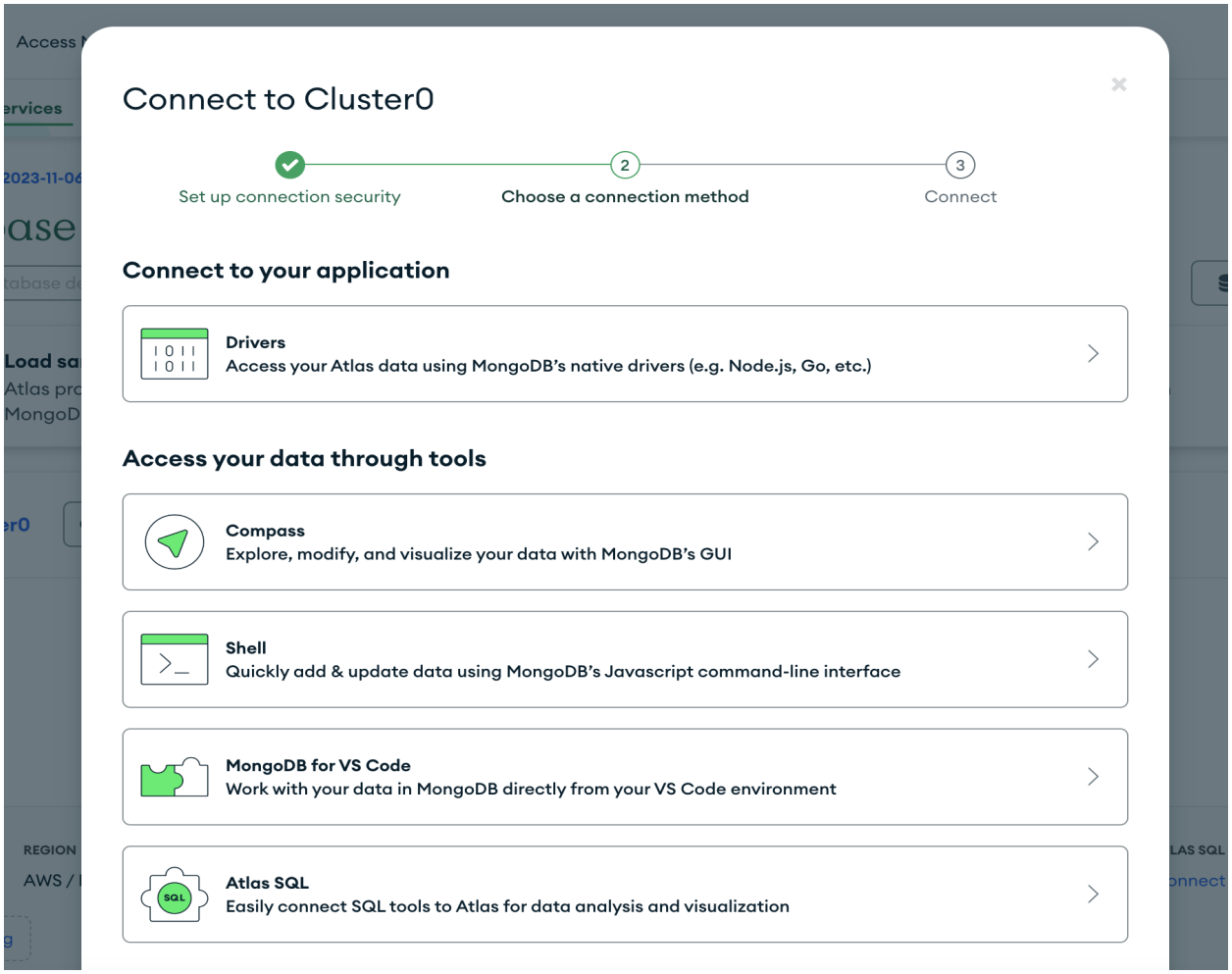
5-5 Remote Connection to Database using Compass

To establish a remote connection, we utilize MongoDB Compass—an effective graphical user interface for querying, aggregating, and analyzing MongoDB data visually. Compass is freely available and open-source, compatible with macOS, Windows, and Linux. Download Compass and refer to the installation instructions for further details.

While on the Atlas MongoDB site, ensure you are within the desired project and cluster. Navigate to the right-side menu, select "Database," and then proceed to click on "Connect."



On the landing page for "Connect," you have the option to choose your preferred connection method. As previously detailed in this documentation, we will be using Compass.



Upon reaching the landing page for the "Compass" option, you'll find various download links for Compass. Below, there are links available for MacOS 64-bit (10.14+) and Windows 64-bit (10+).

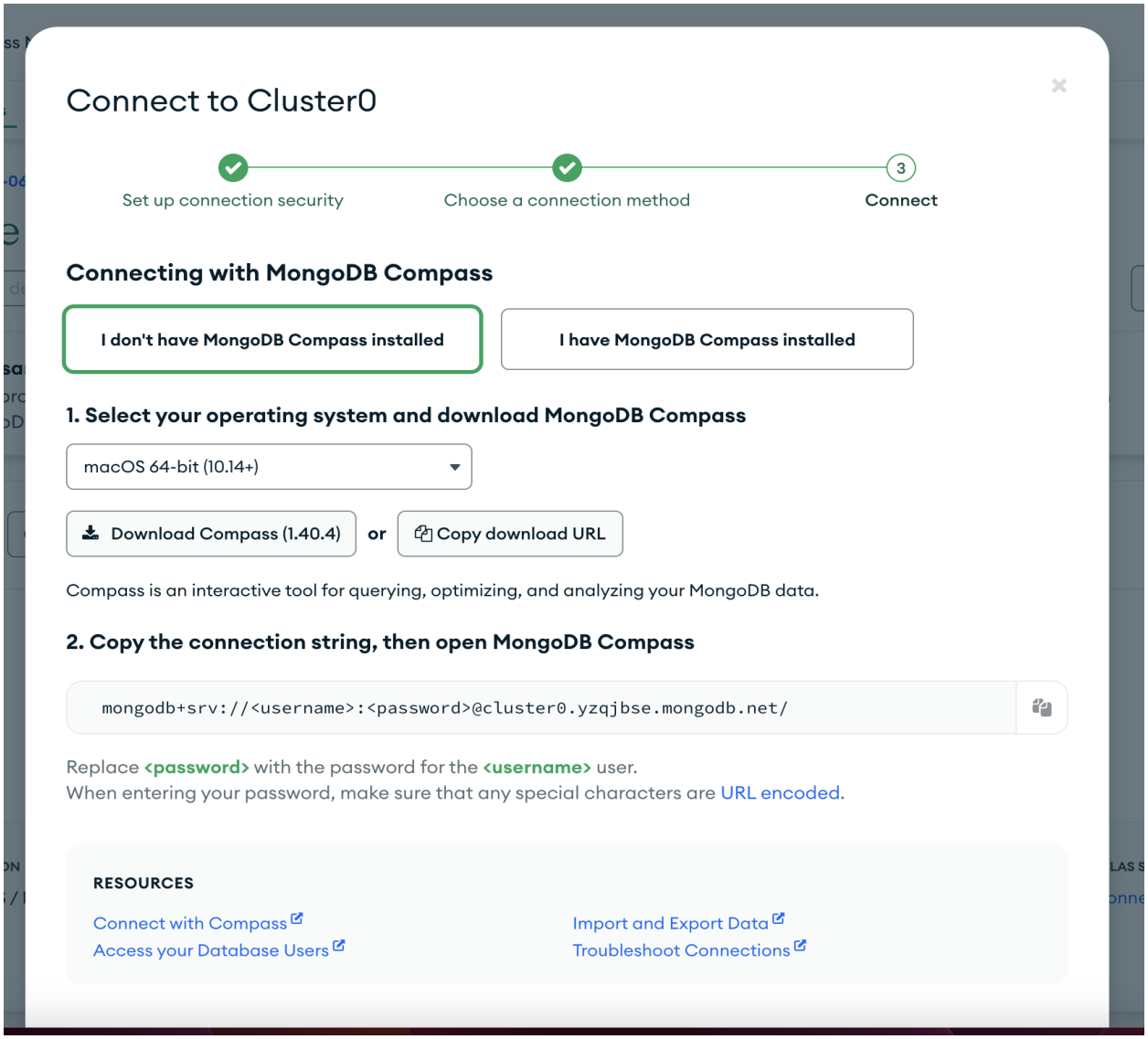
Compass download links:

MacOs 64 bit (10.14+):

https://downloads.mongodb.com/compass/mongodb-compass-1.40.4-darwin-x64.dmg

Windows 64 bit (10+):

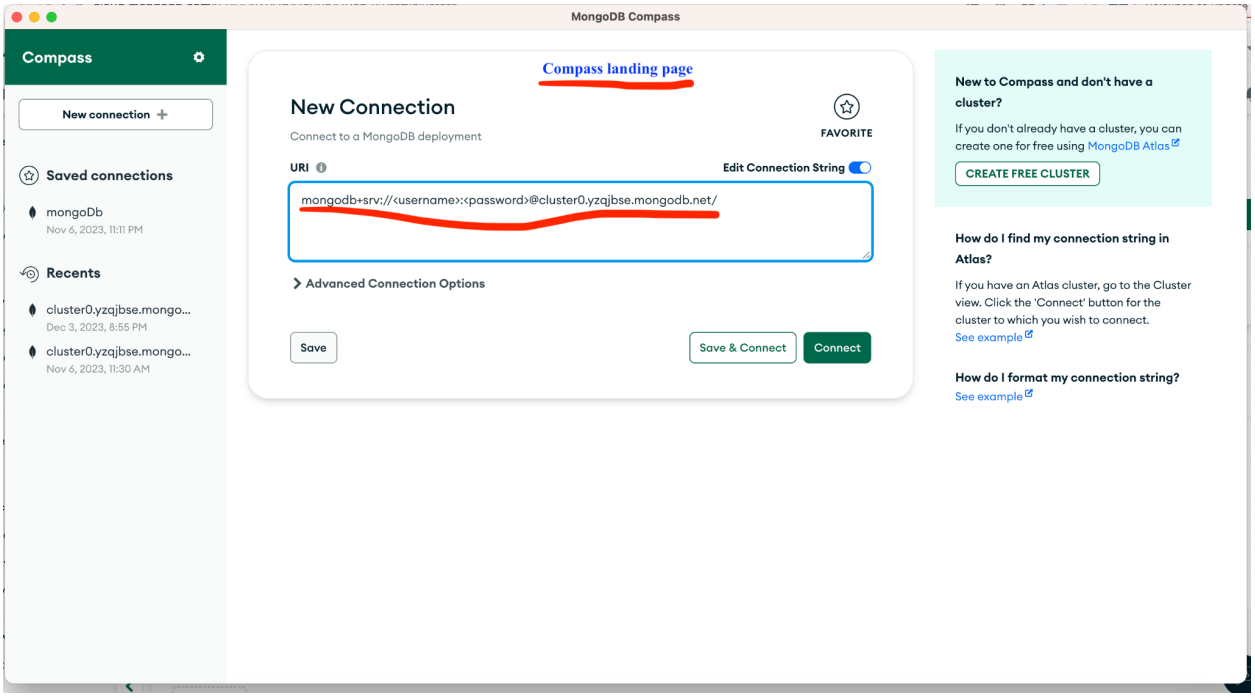
https://downloads.mongodb.com/compass/mongodb-compass-1.40.4-win32-x64.exe



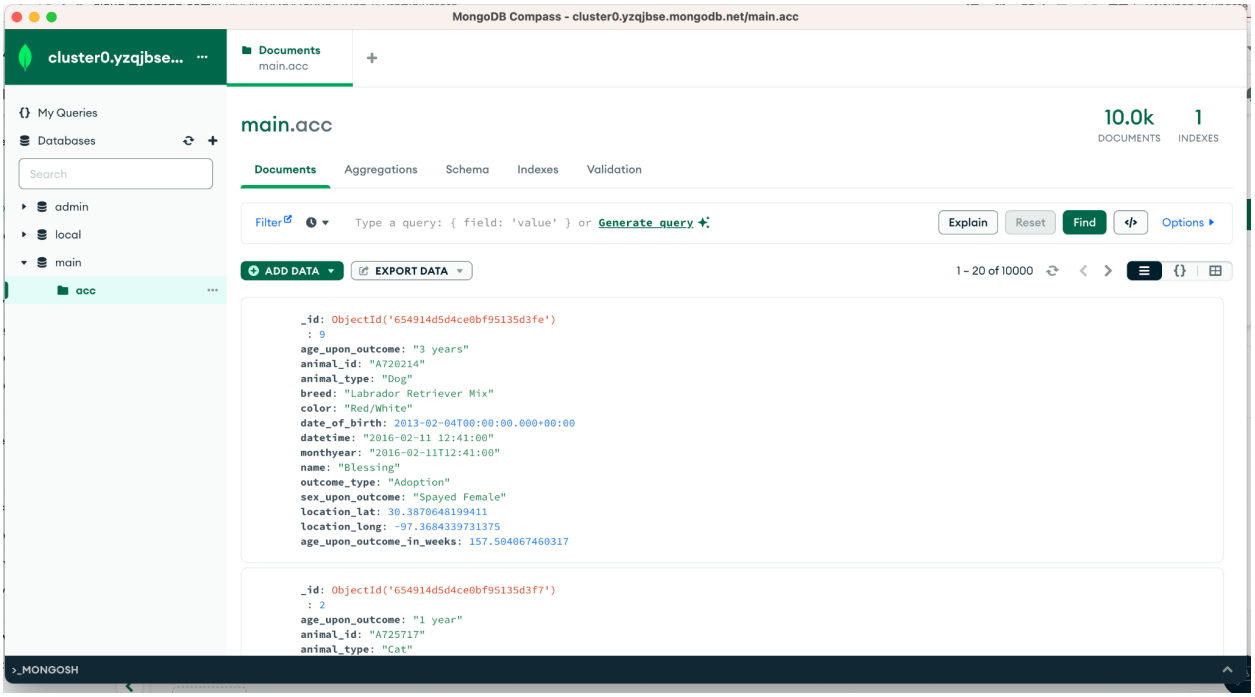
Once Compass is downloaded and installed, open Compass and the following command to connect:

mongodb+srv://<username>:<password>@cluster0.yzqjbse.mongodb.net/

Replace <username> and <password> with your login information (I sent the login information on email)



Once you connect, you should be able to interact with the data



5-6 Benefits of using cloud solution for the database

In this segment, I'll elaborate on the significance of this enhancement and the benefits it offers to end-users, whether they are technical individuals (such as software engineers, architects, or database administrators) or non-technical users (regular users).

5.6.1: Benefits for technical individuals:

Hosting a dataset on a cloud database offers several benefits:

1. Scalability: Cloud databases provide scalable solutions, allowing you to easily scale your dataset storage as your data needs grow. This ensures that you can accommodate increasing volumes of data without worrying about infrastructure limitations.

2. Accessibility: Cloud databases enable remote access to your dataset from anywhere with an internet connection. This accessibility is particularly beneficial for teams spread across different locations or for applications that need to serve users globally.

3. Cost Efficiency: Cloud databases often follow a pay-as-you-go model, allowing you to pay only for the resources you use. This can be more cost-effective than maintaining and managing physical servers, especially for smaller projects or startups.

4. Reliability and High Availability: Cloud providers typically offer robust infrastructure with redundancy and backups, ensuring high availability and data durability. This reduces the risk of data loss and minimizes downtime.

5. Security Measures: Cloud providers invest heavily in security measures, including encryption, identity management, and compliance certifications. This helps enhance the overall security of your dataset, protecting it from unauthorized access or data breaches.

6. Automated Maintenance: Cloud databases often come with automated maintenance tasks, such as backups, software updates, and scaling operations. This reduces the administrative burden on your team and ensures that your database is well-maintained.

7. Integration with Other Cloud Services: Cloud databases can easily integrate with other cloud services, such as analytics tools, machine learning platforms, and serverless computing. This facilitates a more comprehensive and efficient data ecosystem.

8. Faster Deployment: Setting up a database on the cloud is typically quicker than procuring and configuring physical hardware. This allows for faster deployment of applications and reduces time-to-market for projects.

In summary, hosting a dataset on a cloud database provides advantages in terms of scalability, accessibility, cost efficiency, reliability, security, automated maintenance, integration capabilities, and faster deployment, making it a popular choice for many organizations and projects.

5.6.2: Benefits for non technical users:

1. Accessibility and Ease of Use: Cloud databases enable applications to run seamlessly without requiring end users to manage or understand the underlying technical infrastructure. This ensures that users can interact with applications and access data without the need for technical expertise.

2. Reliability and Availability: Cloud databases often provide high reliability and availability, which means end users can access their data whenever they need it. This reliability ensures a consistent and uninterrupted user experience.

3. Data Security: Cloud providers implement robust security measures to protect data. End users benefit from this by having their data stored in a secure environment without needing to actively manage security protocols.

4. Automatic Updates and Maintenance: Cloud databases often handle updates and maintenance tasks automatically. End users don't need to worry about manually updating software or managing server maintenance, ensuring that the application remains up-to-date and available.

5. Collaboration and Sharing: Cloud databases facilitate easy collaboration and data sharing among users. Whether it's a collaborative project or sharing data with colleagues, the cloud infrastructure simplifies these processes, making it user-friendly for non-technical individuals.

6. Scalability without User Intervention: If an application grows, cloud databases can scale resources automatically. This scalability happens behind the scenes, without requiring end users to understand or manage the technical aspects of scaling.

In essence, cloud databases contribute to a user-friendly experience for individuals who may not possess technical knowledge, offering them reliable, secure, and accessible access to applications and data.

5-7: Course objective and skills showcased

This enhancement showcases two course outcomes:

1- 'Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources'

2- 'Employ strategies for building collaborative environments that enable diverse audiences to support organizational decision making in the field of computer science'

Using a cloud database host aligns with the stated goals in the following ways:

5.7.1: Develop a security mindset:

- **Anticipating Adversarial Exploits:** Hosting a database on a cloud platform involves leveraging the security features provided by the cloud service provider. Cloud databases often come with built-in security measures, such as encryption, access controls, and monitoring, which can help anticipate and defend against adversarial exploits.

- **Mitigating Design Flaws:** Cloud providers invest heavily in ensuring the security of their infrastructure. By utilizing a cloud database host, one can benefit from the provider's security protocols, helping to mitigate potential design flaws that might arise in a self-hosted environment.

- **Ensuring Privacy and Enhanced Security:** Cloud databases are designed to prioritize data privacy and security. They offer features like data encryption in transit and at rest, compliance certifications, and regular security updates, contributing to enhanced security and privacy of data and resources.

5.7.2: Employ strategies for building collaborative environments:

- **Building Collaborative Environments:** Cloud database hosting facilitates collaboration by providing a centralized and accessible repository for data. Multiple users, regardless of their physical location, can access and work with the data concurrently. This collaborative environment supports organizational decision-making by ensuring that diverse audiences can contribute to and utilize data in real-time.

- **Enabling Diverse Audiences:** Cloud databases typically offer user management features that allow administrators to define roles and permissions, ensuring that diverse audiences can engage with the data according to their responsibilities. This flexibility contributes to building a collaborative environment where individuals with varying levels of technical expertise can participate in decision-making processes.

In summary, the technical skills required involve a combination of security-focused knowledge, cloud computing expertise, database management proficiency, and a strong foundation in secure software development practices. These skills collectively empower

individuals to design and implement secure, collaborative, and efficient systems in the field of computer science.

6- ePortoflio Link

https://github.com/ilyass-Hm/ePortfolio