Report Hand-In

Using MongoDB with Python

(Restaurant finder App)

+

CST8276-Advanced Database Topics

Professor Taghreed Altamimi

Due: August 2, 2021

By: Group 5

Table of Contents

Group Members:

Abdullahzeki Ilgun	ilgu0001@algonquinlive.com
Farideh Shiran	shir0090@algonquinlive.com
Fargol Azimi	azim0008@algonquinlive.com
Janio Mendonça Junior	mend0152@algonquinlive.com
Marcelo Monteiro da Silva	mont0371@algonquinlive.com
Pardis Feizi	feiz0009@algonquinlive.com

Introduction:

The restaurant finder app has been designed and implemented by group 5 using MongoDB, a NoSQL database system. This project stores a catalogue that includes a list of restaurants with their names and geographical coordinates. The project contains 3 parts since it is following the MVC model (Model, View and Controller).

The view section, which contains the front-end web application, has been programmed using React and Leaflet to bring up a page with all the information. On the front-end side, there is a plot indicating the location of the restaurant. The program allows the user to search for restaurants located nearby with getting input for the radius in kilometers and zip code or the address.

The requests from the user are made by fetching the API to the routes in the backend section of the project. The GET and POST requests are handled by Python using Flask. Moreover, there is a communication with MongoDB Atlas cloud cluster where the data of the restaurants, the catalogue, is stored.

Most of the team members are studying and learning Python now. This project was a terrific opportunity to implement their knowledge into a practical project. On the other hand, non-relational databases like MongoDB are becoming extremely popular in the technology field and companies. The team has decided to work on this project to enhance their knowledge and experience in storing data with this approach. This approach represents the information in JSON-like documents instead of using the conventional row and table format that database management systems usually use.

Problem Description:

WHO: Who cares and who does it affect?

Two groups of people can be affected by the restaurant's finder app:

- Software developers
- People who use this app

Because of the ease of handling unstructured data, development teams prefer MongoDB over relational databases. The ability to scale efficiently is one of the most significant advantages of MongoDB over relational ones. MongoDB provides flexible data models, allowing the database schema to be flexible and evolve by business requirements. For instance, MySQL on the other hand, has a rigid structure that needs extensive engineering to scale. That is why the Mongo DB impresses our team to choose over other DBMS.

One major advantage it has over relational databases is its ability to handle "humongous" amount of unstructured data (Indeed, that's where the name "Mongo" comes from). It moves at a breakneck pace. People are experiencing real-world MongoDB performance because it allows users to query in a more workload-sensitive manner. [1]

Dining out has evolved into a new type of leisure that allows people to broaden their horizons or simply have fun. However, with the increasing number of different restaurants, making the appropriate option is not always straightforward. Alternatively, they may live two blocks from a fantastic restaurant that they will undoubtedly enjoy, but they are unaware of its existence. This application will accept the user's inputs such as:

- People who live in the specified location (based on radius area)
- People who look for the exact name of the restaurant
- People who seek a restaurant by using the zip-code

It has the potential to make many people's lives much easier, and it also has the potential to benefit them much! It is sufficient to state that the restaurant industry's annual sales are increasing. Mobile technologies have a significant impact on several aspects of our lives, and restaurant services are no exception. [1]

WHAT: What is the data aspect of the issue?

This adheres to the BASE (Basic Availability, Soft-state, and Eventual consistency) model. On a single document level, it supports atomic updates. It is conceivable to execute transaction-like semantics across MongoDB. However, this will necessitate complicated implementations.

MongoDB supports atomic modifications. In other words, if you update two values within a document, either both are successfully updated or remain unchanged. Until today, it did not support transactions or ensure data consistency. However, changes in MongoDB 3.6, such as the addition of client sessions and casual consistency, have given application developers more flexibility in fine-tuning the desired consistency the program demands.

The Read Concern and Write Concern parameters in MongoDB provide a customizable consistency approach. A casually consistent session means that the read and acknowledged write operations in the related sequence have a causal relationship that is reflected in their ordering. In a client session, applications must ensure that only one thread executes these activities at a time. For one of our clients, we used MongoDB to improve the scalability of read-write operations. For example, assume our address table included the name and location of the restaurant.

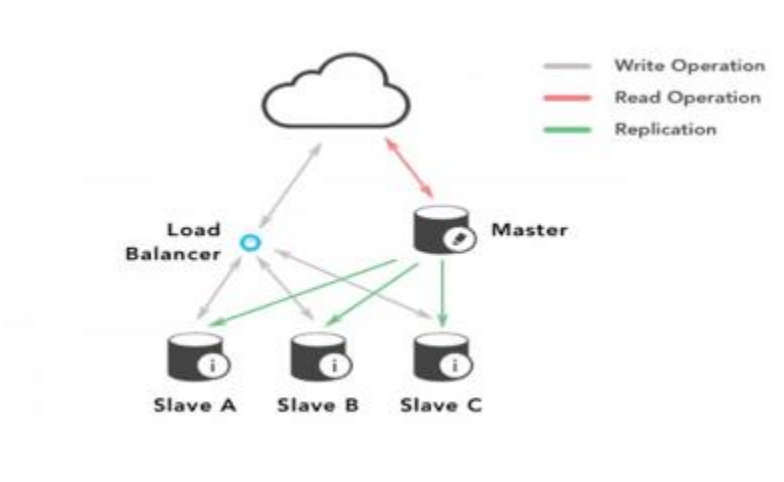
When the menu of the restaurant is entered into the specific restaurants, it is added to the dataset. If we run those two updates separately, one could succeed while the other fails, causing the figures to be uncoordinated. Placing the same updates within them guarantees either success or failure.

WHERE: is this a client-side problem, a replication problem, a server capacity problem, etc.

This is a client-side as well as a replication problem. It is a client-side issue since they cannot properly access the data in a way that is convenient for them, yet the database is at the source of the problem. Both the data and its design must be constructed and inserted in such a way that the data is delivered to the user's benefit.

When a user may enter the incorrect file name or enter characters where a number should have been entered. Alternatively, the client sends an invalid web service request. In other words, the client is not following the terms of the contract when interacting with the application. Furthermore, it occurs when the client's input violates the rules, and the webserver is unsure how to process a request with an abnormal syntax.

A server capacity problem is when without trustworthy data, making any application or environment changes will always result in poor results. Profilers are useful for learning about server applications, but they are typically erroneous since data derived from a single application user may differ dramatically from data derived from dozens or even hundreds of application users.



MongoDB only supports master-slave replication. It makes multiple copies of the data using replica sets. At any point during the process, each member of the replica set will be assigned a primary or secondary role. By default, read/write operations are performed on the primary replica, which is then replicated on the secondary replica.

In addition, the replication strategy in master-slave architecture causes the client to be blocked until the leader obtains a positive answer from all the followers. Waiting for data to be duplicated at each node in a high-read scalable system with thousands of following nodes is not a good user experience.

WHEN: any timing aspects (processes, steps, etc.)

of the most important aspects of the project is time management and setting the timeline for each step of the project. Creation of the schema and defining the data will be the first process of the project. This step occurs with data analysis. Subsequently, activities associated with a schema and relationships will be defined which is the logical design process. In the end, implementation will take place after testing everything is working according to expectations. [2] The steps of the development can be divided into three parts:

- View Layer: This layer is responsible for the user interface. This step requires working with React, Leaflet and to provide a dynamic, user-friendly GUI for searching for various restaurants.
- Controller: This layer provides the connection between the view layer and the database. The business logic is defined in this layer. PyMongo, Flask, and GeoPy will be used in this step.
- Model: This layer is the actual database. Any retrieval or insertion of data happens in this step. This step contains the MongoDB database and the information of different restaurants.

Understanding the complete process aids in estimating the time it will take to construct an app. It entails a series of intertwined processes that every application must go through. We take a look at each of those steps and describe what happens at each one.

- Back-end development

Back-end developers are in charge of creating back-end architecture, which includes the app server and database. Every dynamic application necessitates data synchronization and storage capacity. All these features are dependent on the work of the back-end developer. Back-end development takes 1-2 weeks to complete because it involves working with the app server and database. The data for such service applications is stored entirely on the back-end server.

- Front-end development

The primary responsibility of front-end developers is to design the application's interface. They attempt to recreate the app's appearance as closely as possible to the initial concept. In today's highly competitive industry, this is likely the most critical component of the development process from a business standpoint. The development team can begin parallel work on the front-end element of the product after the server is in development and the database structure is complete. The data from the previously mentioned back-end server will be rendered and organized. From 1 to 2 weeks, front-end developers typically manage the tasks that enable functionality.

WHY: any laws, regulations, or other constraints

This employs role-based access control with a flexible set of privileges. Authentication, auditing, and authorization are among its security features. Furthermore, Transport Layer Security TLS and Secure Sockets Layer SSL can be used for encryption. This ensures that it is only accessible and readable by the intended client.

There are some indexes in MongoDB which enforce some rules. As an example, the unique index makes sure there are no duplicate values in the data. In MongoDB, data coming from INSERT and UPDATE operations will be validated. Data administrators can specify rules for collections with the help of the validator option in MongoDB. [3]

HOW? how does the issue happen and get resolved?

There are a lot of restaurants located around. From chain restaurants to small privately-owned restaurants provide people different options to choose from. Some of the new restaurants have the problem of not being known and shown in the maps and online delivery apps. This results in fewer customers using these restaurants and bringing down the business. When a restaurant gets closed or there is a new restaurant opening in an area, it might take weeks for these applications to be updated. The absence of a structured well-designed database for the restaurants available could result in information not being updated. By using this project, the restaurant app finder will reduce the confusion for the user. This technology could be also used in navigating restaurants in food delivery applications.

This project has a web catalogue of the list of restaurants near an area. MongoDB is a great technology to use with big data as well as being schema-free which has the flexibility to work with various data typed. There have been different technologies used in this project to develop various parts:

- React
- API map plot
- Geopy
- Flask
- MongoDB

Solution Description:

What you will be doing for the solution demonstration and what do you need to build:

According to the problem definition, dining out is one activity that is increasingly enjoyed by people who want to have an enjoyable time, enjoying decent food with family, friends or even alone. However, the problem arises when there are many restaurant choices nearby and people do not even know that they are located around.

The proposal to overcome the issue stated is to develop an application where the user will take inputs like city, zip-code or even a specific restaurant and define the radius of the

search. The application will have a MongoDB database using python to connect the database and JavaScript to show the results to the user. [4]

Components, Script, Database setup, etc.

The Solution demonstration will be done in 5 different steps:

MongoDB:

First, for the database demonstration, the team worked on MongoDB. MongoDB is a NoSQL database that is based on a document database. MongoDB Provides datasets and retrieval of documents from a collection of documents that allows developers to move fast. We create the MongoDB Atlas Cloud Server to save our data that every member has permission to read and write and allowed us to make CRUD operations from anywhere and demonstrate the project. MongoDB has a cluster that includes three servers such as one primary and two secondaries. The reason for having three servers is in case having issues in the primary server, it can failover to secondary servers.

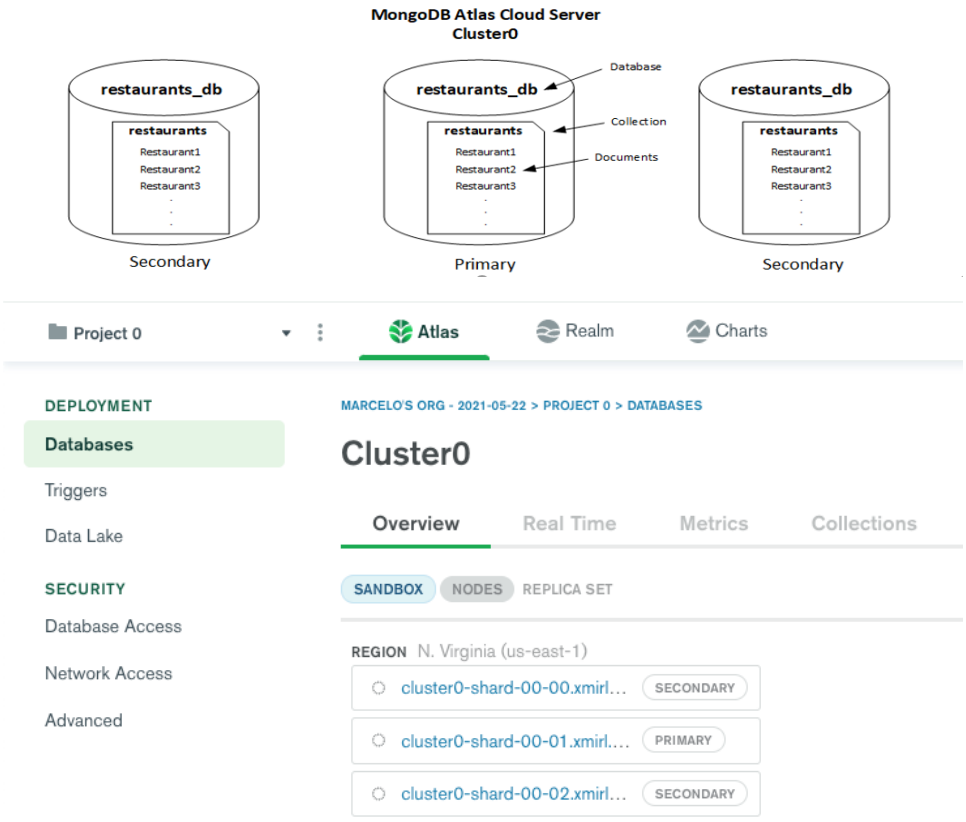


Figure 1: Showing the serves in MongoDB's cluster

Also, we could access the MongoDB compass. It is the client application that we connect the same as the cloud server from our desktop environments.

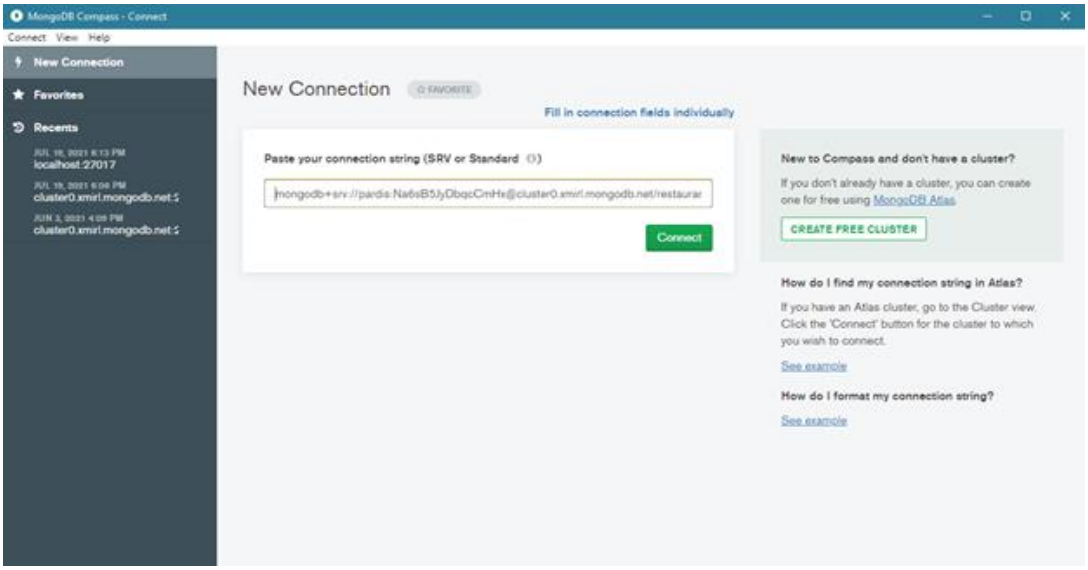


Figure 2: Connecting to MongoDB by using MongoDB compass

MongoDB uses Json-like document with optional schemas, so we import the JSON file that collected all 25K plus documents which is representing the related restaurants' information about various locations of restaurants with ID, restaurant name, and coordinates.


```
{"_id":{"$oid":"55cba2476c522cafdb053add"},"location":{"coordinates":[-73.856077,40.848447],"type":"Point"},"name":"Morris Park Bake Shop"}
{"_id":{"$oid":"55cba2476c522cafdb053ade"},"location":{"coordinates":[-73.961704,40.662942],"type":"Point"},"name":"Wendy'S"}
{"_id":{"$oid":"55cba2476c522cafdb053adf"},"location":{"coordinates":[-73.98241999999999,40.579505],"type":"Point"},"name":"Riviera Caterer"}
{"_id":{"$oid":"55cba2476c522cafdb053ae0"},"location":{"coordinates":[-73.8601152,40.7311739],"type":"Point"},"name":"Tov Kosher Kitchen"}
{"_id":{"$oid":"55cba2476c522cafdb053ae1"},"location":{"coordinates":[-73.8803827,40.7643124],"type":"Point"},"name":"Brunos On The Boulevard"}
{"_id":{"$oid":"55cba2476c522cafdb053ae2"},"location":{"coordinates":[-73.98513559999999,40.7676919],"type":"Point"},"name":"Dj Reynolds Pub And Restaurant"}
{"_id":{"$oid":"55cba2476c522cafdb053ae3"},"location":{"coordinates":[-73.9068506,40.6199034],"type":"Point"},"name":"Wilken'S Fine Food"}
{"_id":{"$oid":"55cba2476c522cafdb053ae4"},"location":{"coordinates":[-74.00528899999999,40.628886],"type":"Point"},"name":"Regina Caterers"}
{"_id":{"$oid":"55cba2476c522cafdb053ae5"},"location":{"coordinates":[-73.9482609,40.6408271],"type":"Point"},"name":"Taste The Tropics Ice Cream"}
{"_id":{"$oid":"55cba2476c522cafdb053ae6"},"location":{"coordinates":[-74.1377286,40.6119572],"type":"Point"},"name":"Kosher Island"}
{"_id":{"$oid":"55cba2476c522cafdb053ae7"},"location":{"coordinates":[-73.8786113,40.8502883],"type":"Point"},"name":"Wild Asia"}
{"_id":{"$oid":"55cba2476c522cafdb053ae8"},"location":{"coordinates":[-73.9973325,40.61174899999999],"type":"Point"},"name":"C \u0026 C Catering Service"}
{"_id":{"$oid":"55cba2476c522cafdb053ae9"},"location":{"coordinates":[-73.96926909999999,40.7685235],"type":"Point"},"name":"1 East 66Th Street Kitchen"}
{"_id":{"$oid":"55cba2476c522cafdb053aea"},"location":{"coordinates":[-73.871194,40.6730975],"type":"Point"},"name":"May May Kitchen"}
{"_id":{"$oid":"55cba2476c522cafdb053aeb"},"location":{"coordinates":[-73.9653967,40.6064339],"type":"Point"},"name":"Seuda Foods"}
{"_id":{"$oid":"55cba2476c522cafdb053aec"},"location":{"coordinates":[-73.97822040000001,40.6435254],"type":"Point"},"name":"Carvel Ice Cream"}
{"_id":{"$oid":"55cba2476c522cafdb053aed"},"location":{"coordinates":[-73.7032601,40.7386417],"type":"Point"},"name":"Carvel Ice Cream"}
{"_id":{"$oid":"55cba2476c522cafdb053aee"},"location":{"coordinates":[-74.0259567,40.6353674],"type":"Point"},"name":"Nordic Delicacies"}
{"_id":{"$oid":"55cba2476c522cafdb053aef"},"location":{"coordinates":[-73.9829239,40.6580753],"type":"Point"},"name":"The Movable Feast"}
{"_id":{"$oid":"55cba2476c522cafdb053af0"},"location":{"coordinates":[-73.839297,40.78147],"type":"Point"},"name":"Sal'S Deli"}
{"_id":{"$oid":"55cba2476c522cafdb053af1"},"location":{"coordinates":[-73.95171,40.767461],"type":"Point"},"name":"Glorious Food"}
{"_id":{"$oid":"55cba2476c522cafdb053af2"},"location":{"coordinates":[-73.9925306,40.7309346],"type":"Point"},"name":"Bully'S Deli"}
{"_id":{"$oid":"55cba2476c522cafdb053af3"},"location":{"coordinates":[-73.976112,40.786714],"type":"Point"},"name":"Harriet'S Kitchen"}
{"_id":{"$oid":"55cba2476c522cafdb053af4"},"location":{"coordinates":[-73.94024739999999,40.7623288],"type":"Point"},"name":"Steve Chu'S Deli \u0026 Grocery"}
{"_id":{"$oid":"55cba2476c522cafdb053af5"},"location":{"coordinates":[-73.96805719999999,40.7925587],"type":"Point"},"name":"P \u0026 S Deli Grocery"}
{"_id":{"$oid":"55cba2476c522cafdb053af6"},"location":{"coordinates":[-73.996984,40.72589],"type":"Point"},"name":"Angelika Film Center"}
{"_id":{"$oid":"55cba2476c522cafdb053af7"},"location":{"coordinates":[-73.9634876,40.6940001],"type":"Point"},"name":"White Castle"}
```


Figure 3: JSON File

After connecting to MongoDB Atlas cloud server, we create the new database which is named restaurant_db, and the collection is named restaurant that contains documents.

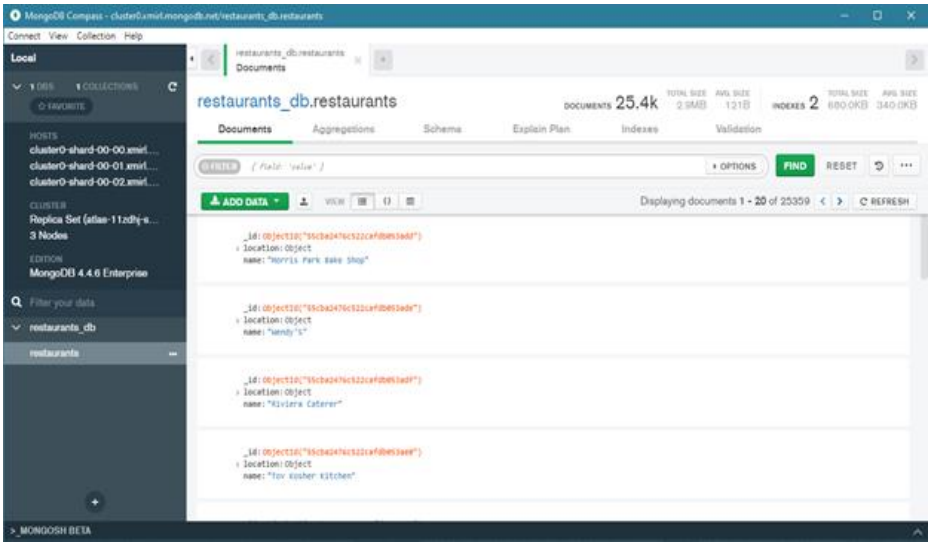


Figure 4: View Data from MongoDB Collection

We created a new index under the restaurant database by configuring the location and 2dsphere. A 2dsphere index supports queries that calculate geometries on an earth-like sphere. 2dsphere index supports all MongoDB geospatial queries.

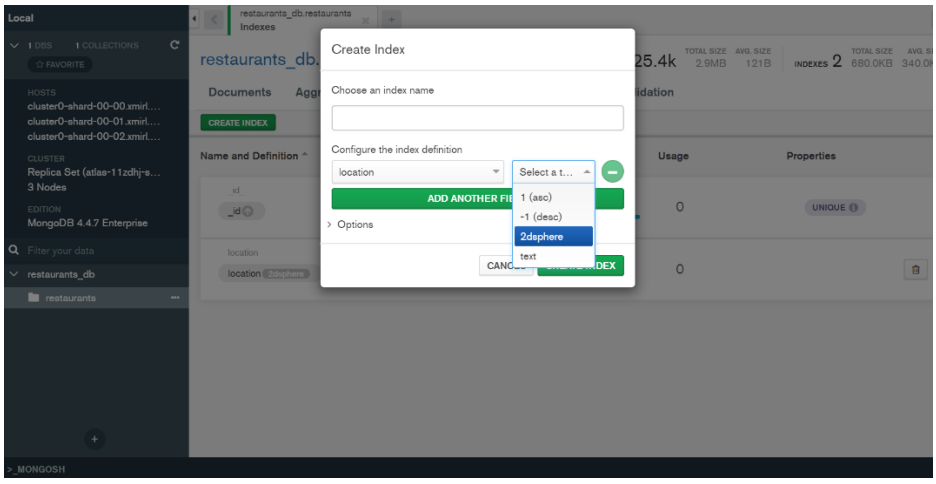


Figure 5: Creating new index based on Location and 2dsphere

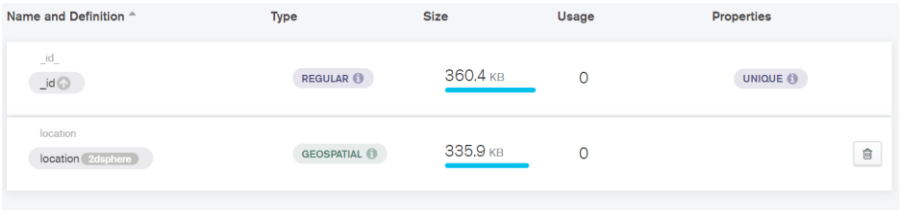


Figure 6: Showing the created index "Geospatial"

The reason of creating the index is allowing us to find the location base on coordinates.

```
results = collection.find(
    {'location': {'$nearSphere': {'$geometry': {'type': "Point",
                                                'coordinates': [float(lon), float(lat)]},
                                  '$maxDistance': radius}},
     'name': {'$regex': restaurant_name, "$options": "i"}},
    {"_id": 0})

return results
```

Figure 7: for making this query to work, we created the index

The connection between backend and database is happening with PyMongo. PyMongo is a library that communicates between the database and the backend. The following python code which is mongodb.py returns a MongoClient Object to be used by other modules. We provided the MongoDB server address and identification credential as parameter in the constructor of the MongoClient object and the client is used by all other modules to connect to our MongoDB database.

```
def connect_to_mongodb(database, credentials_file):
    """
    This function connects to mongodb using the credentials in the credentials file
    and saves the connection in the client variable.
    """
    # Change the scope of client variable to global and calls get_credentials_from_file method
    global client
    get_credentials_from_file(credentials_file)

    # It only try to connect if user and password is not empty or it is just an empty space
    if (username and password != "") and (len(username) and len(password) > 1):
        client = MongoClient(
            f'mongodb+srv://{username}:{password}@cluster0.xmirl.mongodb.net/{database}?retryWrites=true&w=majority',
            ssl=True, ssl_cert_reqs='CERT_NONE')
    else:
        print("User name and/or password is empty, please check the credentials.pwd file")
        sys.exit(1)

    return client
```

We created another function called create_and_populate_database. This function has created the database in our MongoDB server and then populate the database will collection and document from the JSON file that we provided as a parameter.

```python
# Defining database and collection name to connect to mongodb cluster
database_name = "test_db"
collection_name = "test"
# Collection json file to be imported into mongodb database
collection_file = '/../collections/test.json'
# Credentials file path and name
credentials_file = "/../credentials.pwd"

# Connect to mongodb with mongodb module
client = connect_to_mongodb(database_name, credentials_file)
# Defining database to work with
db = client[database_name]
# Defining collection to work with
collection = db[collection_name]


def create_and_populate_database(collection_file):
    """
    This function creates a database and collection in the MongoDB server
    :param collection_file: Json file with all documents to be inserted in the collection
    """
    try:
        # Reads the json file and save the content in the variable documents
        with open(os.path.dirname(__file__) + collection_file, ) as file:
            documents = file.readlines()
            file.close()
    except FileNotFoundError as error:
        print(error)
        sys.exit(1)

    # Loop through the documents variable and convert/loads each string line to json and insert in the collection
    counter = 0
    for document in documents:
        counter += 1
        collection.insert_one(bson.loads(document))
        print(f"Progress: {int(counter / len(documents) * 100)}%")
```

Flask:

Flask is a Python web framework that allows developers to customize how users interact with data. In the restaurant finder app, flask is in the controller that works in the backend and is responsible for the API server.

We have one python flask file (flask_api_routes.py) in the project. This file acts as a coordinator between model and view. Therefore, the flask will communicate between the backend and frontend.

We will have the python flask file with one route. The /api/restaurants/ accepts GET and POST method and it takes the following parameters from the frontend: Restaurant Name, Zipcode and Radius. PyMongo is used to connect to the MongoDB using python, a Flask backend and Geopy to process the query and get the results from the database and frontend to plot the nearby restaurant results on the map

A GET method requests data from the specified resource, so the GET message is sent, and the server returns the data, and a POST method sends data to a server to create/update a resource. Hence, it sends form data to the server and the data received by the POST method is not cached by the server.

We have 2 methods in the flask: restaurant_finder() and start_server(). The restaurant_finder() is known as the main method. This function accepts GET requests. If /API/restaurant is put on the browser of address, frontend is started and all information will be passed, the frontend sends all parameters with the GET method to the backend as well as restaurant_finder function will be executed. The flask saves the variables: the restaurant name,

zip code and radius, and then call the controller. Actually, it calls the find_resturant function from the controller and passed 3 variables. Finally, it returns JSON data with a list of restaurants to the frontend, see the figure below.

```
16  @app.route("/api/restaurants/", methods=["GET", "POST"])
17  def restaurant_finder():
18      """
19      This function accepts GET and POST requests, saves name of the restaurant, zip code and radius
20      and calls the controller to query de database and returns a list of restaurants to the front-end
21      """
22      restaurant_name = request.args.get('restaurantName')
23      zip_code = request.args.get('zipCode')
24      radius_in_km = request.args.get('radius')
25
26      results = controller.find_restaurants(restaurant_name, zip_code, radius_in_km)
27
28      return jsonify(results)
```

From the controller file, the find_resturants method finds all documents using the restaurant name, zip_code that will be used by Geopy to find relatively precise coordinates and radius in kilometres. Since MongoDB uses the distance in meters, we use the variable in line 75 (figure below) to convert distance from kilometres to meters. In addition, the latitude and longitude coordinates give a precise location. Therefore, we call Geopy and access to zip-code by the variable of coordinates in line 76 (the figure is shown in the following).

```
65
66  def find_restaurants(restaurant_name, zip_code, radius_in_km):
67      """
68      This method finds all documents using restaurant name, radius near lat and lon
69      :param restaurant_name: Name of the restaurant Ex: {"name": "Wendy'S"}
70      :param radius_in_km: Radius in km used to find restaurants
71      :param zip_code: Zipcode used by GeoPy to discover the coordinates
72      """
73      restaurant_list = list()
74      meters_per_km = 1000
75      maximum_distance_in_meters = int(radius_in_km) * meters_per_km
76      coordinates = geopy_find_coordinates(zip_code)
77      lat = coordinates.raw['lat']
78      lon = coordinates.raw['lon']
```

The second function in the flask file is start_server. It starts flask server by 2 parameters: server's IP and debug mode (that is true or false). Furthermore, it runs backend by calling this function.

```
31  def start_server(host_ip, debug_mode):
32      """
33      This function starts flask server to run the back-end
34      :param host_ip: IP of the flask server
35      :param debug_mode: Boolean to activate or not flask debug mode
36      """
37      app.run(host=host_ip, debug=debug_mode)
```

When the program is run by the main method. It calls the start_server function from the controller and starts backend.

```
10  from controller import flask_api_routes
11
12  # Run the program
13  if __name__ == "__main__":
14      # Start the back-end
15      flask_api_routes.start_server("127.0.0.1", True)
16
```

Geopy:

Geopy is a Python client for a variety of popular web-based geocoding services. Using third-party geocoders, Geopy makes it simple for Python developers to get the coordinates of addresses, cities, countries, and landmarks all over the world. The project uses Nominatim which will define latitude and longitude. The Nominatim uses OpenStreetMap which has free access.

When we enter the zip code, it finds coordinates (latitude and longitude) by the geopy_find_coordinates function and passes the zip_code parameter as is shown in the figure below. The geo_locator is a default configuration for Geopy that uses Nominatim as a geolocation service. The geo_locator function will be called from Geopy and set up the information. Furthermore, the geocode pass the zip code and also, we need to say which country the zip code belongs to that we consider the USA as default.

If a user enters a zip code that has not been defined, the coordinates will be none. Because Geopy will not find the zip code. Hence, we set a zipcode from Manhattan, Newyork for this case. When the entered zip code is found, then it saves the variables as a dictionary. After finding it, the Geopy returns the dictionary.

```
49  def geopy_find_coordinates(zip_code):
50      """
51      This method finds the coordinates (latitude and longitude) of a zip code using GeoPy
52      :param zip_code: Zip code used to search the coordinates by GeoPy
53      """
54
55      if zip_code != '':
56          geo_locator = Nominatim(user_agent='myapplication')
57          coordinates = geo_locator.geocode(zip_code, country_codes="US")
58          # If geopy cannot find the coordinates for the zip code we set a default zipcode from Manhattan New York
59          if coordinates is None:
60              coordinates = geo_locator.geocode("10019", country_codes="US")
61              return coordinates
62          else:
63              return coordinates
```

In addition, we have the coordinates from the find_resturants method as same as return from Geopy as is shown in the figure below. Then, we can save coordinates raw values from latitude and longitude variables. Hence, it can employ how many kilometres the restaurant is.

If a user does not pass the restaurant name, the method of query_resturants_by_location search restaurants based on location and radius that is shown in figure 6. Otherwise, it executes the query_restaurants_by_name_and_location function from the model and so we send the collection from MongoDB. When we call a restaurant by name and location, it receives data from the model.

```
80      if restaurant_name == '':
81          restaurants = model.query_restaurants_by_location(collection, maximum_distance_in_meters, lat, lon)
82      else:
83          restaurants = model.query_restaurants_by_name_and_location(collection, restaurant_name,
84          maximum_distance_in_meters, lat, lon)
85
86      # Loop through the cursor and print each restaurant found and prints
87      for restaurant in restaurants:
88          restaurant_list.append(restaurant)
89          print(restaurant)
90
```

From the model file, the find function tries to find the location. Line 71 in the following figure represents the function from MongoDB, it includes the functions that calculate distance using spherical geometry: show the zipcode as the centre point on the map, and max distance: indicate maximum distance from centre point that is called the radius. For the name of the restaurant, we use the restaurant name with the regex function from MongoDB. The regex function helps to find the restaurant by part of the name. It returns information to the map.

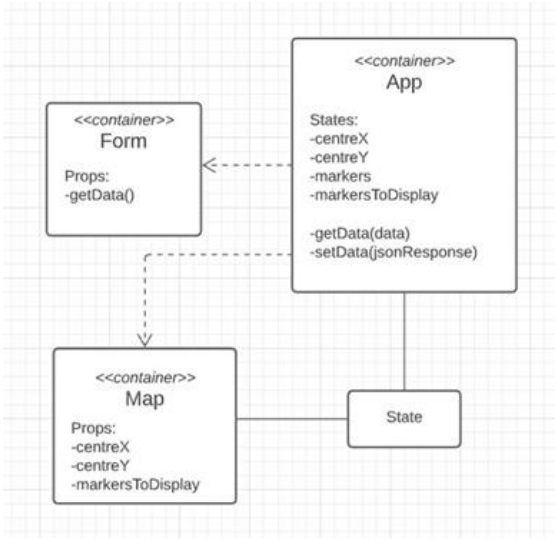
```python
def query_restaurants_by_name_and_location(collection, restaurant_name, radius, lat, lon):
    """
    This method search restaurants in the database based on name, radius and location
    :param collection: MongoDB collection used to search the restaurants
    :param restaurant_name: Name of the restaurant, it does not need to be exact
    :param radius: Max distance in meters to search restaurant around
    :param lat: Latitude of the zipcode
    :param lon: Longitude of the zipcode
    """
    results = collection.find(
        {'location': {'$nearSphere': {'$geometry': {'type': "Point",
                                                    'coordinates': [float(lon), float(lat)]},
                                      '$maxDistance': radius}},
         'name': {'$regex': restaurant_name, "$options": "i"}},
        {"_id": 0})

    return results
```

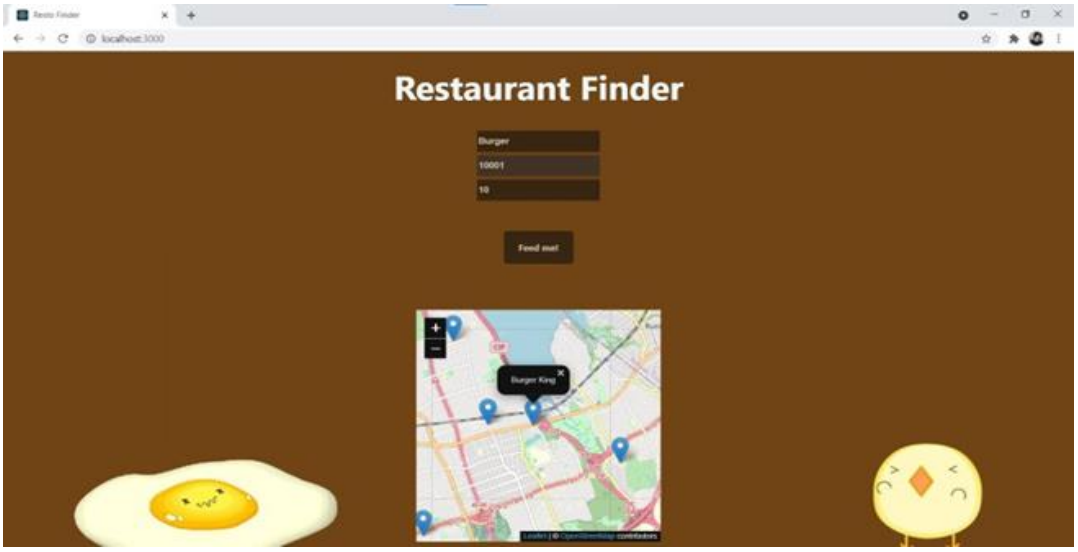
React:

React is a library for developing re-usable components for the front end of an application instead of writing HTML and vanilla JavaScript. This library is open-source, and it is used to build interfaces to interact with the user. This part of the project is the view layer that gets the data from the model since the project is using the MVC architecture. React can manipulate data by using state hooks dynamically without reloading the page which is a bonus for providing a better user experience.

React works as the client-side of the project. This part manages the visual form and the map that the user interacts with. We used React in this project because of the simple structure, and the implementation of the Leaflet library that provides dynamic, customizable maps. [2]



Below is the interface created by React. The user inputs the data in the fields and react will send the data to the back end to get the coordinates and information of the restaurants.



Everything starts with the App module. Modules are being used instead of classes. The program gets the data from the backend via API routes using the URL. By using the data from MongoDB and fetching the data the program sends it as the response afterwards.

```
const getData = (data) => {
  fetch(`/api/restaurants/?restaurantName=${data.name}&zipCode=${data.zipCode}&radius=${data.radius}`, {
    method: 'GET',
  }).then(res => {
      return res.json()
  }).then(jsonResponse => {
      setData(jsonResponse)
      //Looping through all the elements of the markers array, and creating the physical markers
      //using the pre-made leaflet component with JSX.
      console.log(jsonResponse)
      setMarkersToDisplay(markers.map((marker) => {
        return <Marker position = {[marker.location.coordinates[1], marker.location.coordinates[0]]}>
                  <Popup> {marker.name} </Popup>
                </Marker>
      }))
  }).catch((error) => { //If there's no results gotten, error message is displayed on the screen
    console.log(error)
    const node = document.createTextNode(error);
    document.querySelector(".error").appendChild(node)
  })
}
```

Besides the main App component, there are two components in this part of the project that the App component holds: Map and Form components.

The form component gets the data and sends the data to the backend using a POST request through, again, fetch API. There is an already provided useful form of React so that it is used to make the form submission and error handling easier. We put some errors for validating the user input. Postal code and keywords are required to submit the form.

```
const Form = ({getData}) => {

  const {register, handleSubmit, formState: { errors} } = useForm()

  return (
      <>
      <form className='form' onSubmit={handleSubmit((data) => {getData(data)})}>
        <input type="text" name='name' placeholder='Keywords' {...register('name', {required: true})}/> <br />
        {errors.name && <Tooltip/>}

        <input type="text" name='postal-code' placeholder='Postal Code' {...register('zipCode', {required: true})}/> <br />
        {errors.zipCode && <Tooltip/>}

        <input type="number" name='radius' placeholder='Radius' {...register('radius', {required: false})}/> <br />
        {errors.radius && <Tooltip/>}

        <input className='button' type="submit" name='submit' value='Feed me!'/>

      </form>
      </>
  )
}
```

For the focus of the application, the map, a well-known library called Leaflet.js's React implementation was used rather than drawing the map from scratch. In the map component, centre x and y props that are passed from the App component are used with the map's "flyTo" method, so when the user selects a restaurant, the map will be centered to that position with a flying effect. These coordinates are being sent to React from the back end using Geopy in the main component. The property "markersToDisplay" is an array indicating the location of different restaurants on the map.

```
const Map = ({centreX, centreY, markersToDisplay}) => {
  const [map, setmap] = useState(null);

  //Simple statement to change the view of the map
  if (map) {
    map.flyTo([centreY, centreX]);
  }

  return (
        <div className="map">
          <h2></h2>
            <MapContainer
              whenCreated={setmap}
              center = {[0, 0]} //Default centre, can be changed
              zoom={13}
              className="static-map"
              {...interactionOptions}
            >
              <TileLayer
                attribution='&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
              />
              {markersToDisplay}
            </MapContainer>
        </div>
  );
};
```

There are some options for the map that can be set to true. As an example, in the picture below, when the user double clicks on the map it will zoom in. The other option that is set to true is dragging. This means that the user has the option to drag and move the map to different directions to see the restaurants around the area selected.

```
12    iconRetinaUrl:
13      "https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.3.1/images/marke
14    iconUrl:
15      "https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.3.1/images/marke
16    shadowUrl:
17      "https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.3.1/images/marke
18  });
19
20  const interactionOptions = {
21    zoomControl: true,
22    doubleClickZoom: true,
23    closePopupOnClick: false,
24    dragging: true,
25    zoomSnap: true,
26    zoomDelta: true,
27    trackResize: true,
28    touchZoom: true,
29    scrollWheelZoom: false,
30  };
31  /**
```

Lessons Learned

Risks that need to be mitigated (look at your list of known risks)

Developing web applications can become complex and time-consuming when building and maintaining several different technologies that needed to be mitigated for future work.

Considering lighter weight options designed to reduce complexity and time-to-production for our application can result in a more flexible and scalable solution. As a micro web framework built on Python, Flask provides an extensible way for developers to grow their applications through extensions that can be integrated into projects. To continue the scalability of a developer's tech stack, MongoDB is a NoSQL database that is designed to scale and work with frequent changes.

One main risk still could be using the free MongoDB atlas cloud server. Although there are three replicas in the MongoDB server for having errors to pass over to another server, there is still no guarantee of availability.

MongoDB handles real-time data analysis in the most efficient way for big data, since the database is document-based and fields have been embedded, very few queries can be issued to the database to fetch a lot of data. [2]

The project is using a geocoder named Nominate which is free but there is no guarantee that the coordinates retrieved are accurate. That said, the project could be updated to use a more robust geocoder that has more accurate coordinate ate (Latitude and Longitude), for example, Google geocoder. https://algonquinlivecom-my.sharepoint.com/:w:/r/personal/cumm0082_algonquinlive_com/_layouts/15/doc2.aspx?sourcedoc=%7B095a8f3a-f52f-4cce-8786-a9b72f07b2ef%7D&action=edit

We can change the structure of restaurants documents and data over time, which results in a flexible system that allows you to quickly adapt to requirement changes without the need for a complex process of data migration. However, the trade-off in changing the structure of new documents is that exiting documents become inconsistent with the updated schema. So, this is a topic that needs to be managed with care.

[2]

Overall Experience

While working as a computer programmer student, many times we are showing a visual application that demonstrates how data that we are using works while using web apps. We developed a restaurant finder app by using MongoDB, Flask, Leafletjs. This application takes inputs like restaurants name, city or Zip-code and radius from the user and gets all the nearby restaurants within the specified radius. We used PyMongo to connect the MongoDB and flask backend to process the query and get the results. We had so many challenges with the deadlines and creating the web page (Front-end) by using the react. We were able to finish it and demoed the project.

Work Plan:

(GANTT CHART – Deliverables)

Member	Report Writing Responsibilities (Week 2 - 5)	Solution Building Responsibilities (Week 5-10)	Presentation Responsibilities (Week 11-12)	Assigned hours for each member per week
Apo	front end React Research and solution description	React	React demo	10 hours
Janio	Back-end GeoPy research and solution description	Geopy	GeoPy demo	10 hours
Marcelo	Back-end MongoDB research and solution description	MongoDB	Introduction and overview demo	10 hours
Pardis	Problem description	MongoDB	MongoDB demo	10 hours
Fargol	Back-end API map for react research And solution description. Gantt chart	Map API	API map and summary demo	10 hours
Farideh	Back-end flask research and solution description	Flask	Flask demo	10 hours

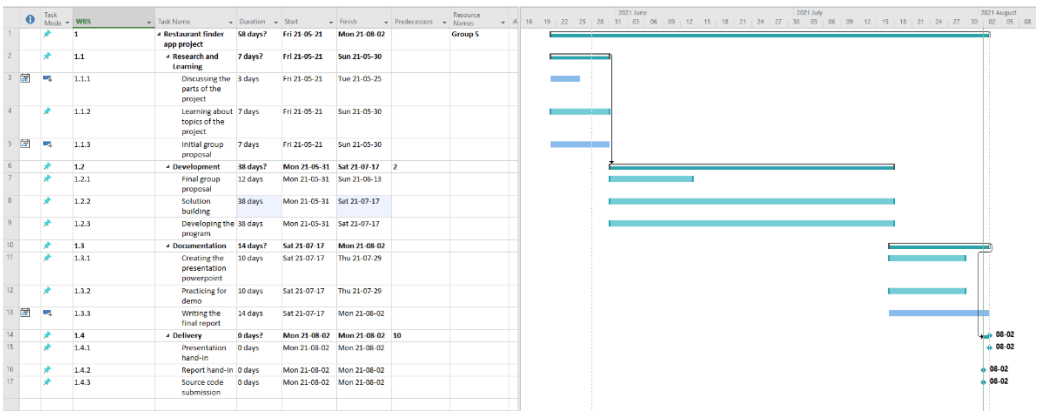


Figure 1: Gantt Chart overview

Works Cited

[1] "MongoDB vs MySQL: A Comparative Study on Databases," 23 November 2017. [Online]. Available: https://www.simform.com/mongodb-vs-mysql-databases/. [Accessed 13 June 2021].

[2] A. W. &. N. ENG, "Chapter 9 Integrity Rules and Constraints," [Online]. Available: https://opentextbc.ca/dbdesign01/chapter/chapter-9-integrity-rules-and-constraints/. [Accessed 27 May 2021].

[3] "Restaurant Finder Apps: How to Build a Similar App?," 21 July 2017. [Online]. Available: https://www.finsmes.com/2017/07/restaurant-finder-apps-how-to-build-a-similar-app.html. [Accessed 27 May 2020].

[4] A. WATT, "Chapter 13 Database Development Process," [Online]. Available: https://opentextbc.ca/dbdesign01/chapter/chapter-13-database-development-process/. [Accessed 27 May 2021].

[5] "Building a Web app using Python and Mongodb," 25 September 2019. [Online]. Available: https://kanoki.org/2019/09/25/building-a-web-app-using-python-and-mongodb/. [Accessed 27 May 2021].

[6] "MongoDB vs MySQL: A Comparative Study on Databases," 23 November 2017. [Online]. Available: https://www.simform.com/mongodb-vs-mysql-databases/. [Accessed 13 June 2021].

[7] N. Pandit, "What And Why React.js," 10 Feb 2021. [Online]. Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/. [Accessed 2 August 2021].

[8] O. B. Henry, "Factors to Consider When Choosing MongoDB for Big Data Applications," 23 October 2019. [Online]. Available: https://severalnines.com/database-blog/factors-consider-when-choosing-mongodb-big-data-applications . [Accessed 2 August 2021].

[9] L. P. Ramos, "Python and MongoDB: Connecting to NoSQL Databases," 1 March 2021. [Online]. Available: https://realpython.com/introduction-to-mongodb-and-python/. [Accessed 2 August 2021].

[10] Jenkov, J. (n.d.). *Error causes, types and reactions*. Tutorials. http://tutorials.jenkov.com/exception-handling-strategies/error-causes-types-reactions.html.

[11] Gautam, N. (n.d.). *5 common types of client side errors*. Annexal. https://annexal.com/5-common-types-of-client-side-errors/.

[12] Small, I. (2005, February 7). *Tackle java server capacity problems*. InfoWorld. https://www.infoworld.com/article/2071839/tackle-java-server-capacity-problems.html.

[13] Gupta, P. (2020, April 23). *System design: Database Replication (part 1)*. Medium. https://medium.com/@pulkitent/system-design-database-replication-part-1-3dcf4a300db1.

[14] *App development timeline: How long does it take?: Existek blog*. RSS. (2020, November 11). https://existek.com/blog/app-development-timeline-how-long-does-it-take.