**Data Storage and Persistence**

Our group (Tour.js) has decided to use a NoSQL database structure for our application, with MongoDB being the top database program of choice. Below is an explanation of why our group is choosing MongoDB as well as how we plan to implement it in our project.

**Why MongoDB?**

**Flexible Schema Design**

MongoDB is great for having a flexible schema design, meaning that data does not have to follow a rigid structure. Considering that a good portion of our application’s userbase will be local Albertan businesses, this would be great for accommodating their profiles. Different clients will have different pieces of information they would like to include on their pages. For example, a restaurant may want to store their menus while a museum may want to share new exhibits. MongoDB’s use of BSON files will allow for this flexibility of data collection, whereas a SQL database would not allow different data fields for each business.

**Geospatial Data**

The History Destination Application itself is meant to help find local tourist business based on their location on the map. One major benefit of MongoDB is that it is great at storing geospatial data, which is one of the main components of our location-based tourist app. MognodDB does this by enabling efficient proximity searches using GeoJSON and specialized indexes, helping the user find relevant data within a specific proximity quickly and efficiently.

**Scalability**

Finally, MongoDB is great in terms of scalability. Considering the app’s planned expansion beyond Albertan tourist markets, MongoDB would be perfect for handling growing data without performance degradation, as its use of horizontal scaling through sharding would help support the app’s growth.

**Implementation**

**Database Structure**

The main collections that we would need to have would include users, business clients, destinations, reviews, subscriptions, and analytics. We can embed the documents within these collections for data that is frequently accessed together (e.g., business hours within business profiles). This will ensure optimal database performance. For many-to-many relationships we may use references instead of embedding as documents to have a 16MB size limit in MongoDB and the data may change frequently. For example, business listings and user reviews would be better suited for reference relationships.

Additionally, businesses will have the option to upload media files (pictures at the start and then move on to videos and audio). What our group plans to do is use MongoDB to store image URLs/references while the actual media files will be managed in Cloudinary, a company that provides loud media management services for websites and apps and integrates MongoDB into its platform. This will ensure optimal database performance and high-quality media integration.

**Data Security**

We will take advantage of Client-Side Field Level Encryption (CSFLE) for sensitive information like user data and payment information. This would prevent MongoDB from accessing this data in an unencrypted form. Additionally, we will want to use frequent automated backups to prevent data loss in our application.

For our authentication process, it will be completed through a third-party (most likely Google Firebase) as MongoDB itself does not have a separate authentication service for logging in. Additionally, for media, our group would benefit from Cloudinary’s built-in security features such as signed URLs, access control and HTTPS delivery.

**Performance Optimization**

Similar to an SQL database, one method that we can use to enhance performance on MongoDB is to use indexing on frequently queried fields such as username, location, and business categories. Indexing is great at organizing information so that the database can quickly find relevant results instead of going through each record to see if there’s a match.

Additionally, we would offload media storage and delivery to Cloudinary, allowing MongoDB to focus on structured database management.