**National Park Meetup**

**Problem Statement**

Design an application which tracks mobile device users as they move through a set of known locations, for example, all [National Parks](https://www.latlong.net/category/national-parks-236-42.html).

A National Park Meetup, Spring Boot application, named **national-park-meetup** was created to track mobile device users as they move through a set of known National Park locations.

This data would be sent to your application in near real-time with a user and location tuple.

This data is simulated in **national-park-meetup** project using a **ScheduledUserLocationTask** service module to send periodic user location updates once every 5 minutes. A stagnation flag of **90%** is set to keep users at their current location. Otherwise, only **10%** of users have a chance to move to a new national park location. These parameters can be altered for different results.

The level of design should be sufficient for passing off to a more junior engineer for implementation.

A junior engineer can continue to build from the 3 implemented services:

1. **UserService** – User management service. Handles user-related actions such as creating, updating, deleting, and retrieving users.
2. **LocationService** – Location management service. Handles park location and user location related actions such as creating, updating, deleting, and retrieving park locations and user locations.
3. **ProximityService** – Proximity service. Determines the proximity of nearby users given a current user’s location found with the user’s mobile number and a given radius in kilometers.

The National Park Service REST API is also documented using Swagger API documentation tools and is available at `http://localhost:8070/docs/index.html when application is executed.

Basic services

Requirements are as follows:

**User Needs**

1. Where is the user currently?

The **national-park-meetup** project's REST API provides a GET request endpoint, “http://localhost:8070/api/location/current?mobileNumber=”, which allows you to retrieve a user's current location at a national park by entering their mobile number as a request parameter.

1. What other users are nearby?

The **national-park-meetup** project's REST API provides a GET request endpoint, “http://localhost:8070/api/proximity/nearby?mobileNumber=&radiusKm=”, which allows you to retrieve all nearby user locations at a national park relative to a user’s location found using the mobile number and radius (in kilometers) request parameters.

**Additional Requirements**

* A user can only be at exactly one location at a given time.

This requirement is understood.

* A user can only be at a location for a period of 1 hour. If the user has not moved to a new location within that time frame, they shall be placed in an inactive state.

The **ScheduledUserLocationTask** module also periodically checks the last time a user’s location was updated. If the user’s location last update plus an hour is less than the current time, then the user’s status is updated to the user status enumeration value INACTIVE.

* The application must use a non-volatile storage engine.

The **national-park-meetup** application stores its data in an H2 database located at **<ROOT\_INSTALL>/data/db/npmeetup\_db** on the file system.

When the application starts, excel files are read to populate the database tables **users**, **location**, and **user\_location**. Spreadsheet data will be used to ingest data only if the tables are empty.

Please note that the H2 database file **npmeetup\_db** will be deleted and then recreated within **<ROOT\_INSTALL>/scripts/windows/dev/test-npmeetup.bat** script to ensure successful execution of functional and unit tests.

**Expected deliverables**

* Design documentation for the application. This should be provided as pdfs, pngs, and/or jpegs.

Please reference this document, <ROOT\_INSTALL>/ReadME.md, and Swagger REST API available at <http://localhost:8070/docs/index.html> for further documentation.

* Implementation for some portion of the application. It is up to you which portion you implement. One suggestion is the user management logic, but feel free to implement another area if it would be more interesting to you. Your implementation should be provided as a zip file containing your project repository (using the VCS of your choice). The zip file should include any build files and non-publicly available libraries required to build your solution.

I implemented the 3 core services to address the questions in this document. Please see above for more details. I will zip the entire national-park-meetup project and send it as the deliverable.

* (Optional, but strongly encouraged) verification of your implementation. This could be unit tests, functional tests, etc. which are included in the project zip file discussed above.

Unit tests and functional tests can be executed using the bat script **<ROOT\_INSTALL>/scripts/windows/dev/test-npmeetup.bat.**

**Time frame**

In order to respect your time, we have tried to constrain this problem to something that could be adequately addressed in about four hours. You may use this time however you wish, but here is a suggested breakdown:

* Design - 2 hours
* Implementation - 1 hour
* Test - 1 hour

**Ground Rules**

* You are free to communicate your design in whatever way you feel is effective, but please provide it in a format that does not require specific tools. The following file formats are preferred: pdf, jpeg, or png

This rule is understood.

* You may utilize any mechanism for feeding input into your solution (for example, using hard-coded data within a unit test or a public data feed).

I used simulated data in excel files located at **<ROOT\_INSTALL>/src/main/resources/xlsx** to initialize the database.

* If warranted, we highly recommend using a unit testing framework such as JUnit.

I used **Mockito** and **JUnit Jupiter** to create my unit and functional tests.

* Additionally, you should use the build and version control (VCS) tools of your choice. Please include all build files and VCS files when submitting the solution.

I used Maven as the build tool to compile the **national-park-meetup** Spring Boot project. Detailed build instructions can be found in README.md.

* Please refrain from sharing your design and solutions on public repository hosts like GitHub and SourceForge. If you are having challenges figuring out a suitable method to submit your solution, ask your interviewers.

This rule is understood.

* You may use other outside resources such as search engines, frameworks, library documentation, or CTI personnel. Please refrain from asking other developers or friends for help solving the coding problem.

This rule is understood. I consulted ChatGPT to expedite project progress. The main area of help I received from ChatGPT was creating the ProximityUtility.calculateDistance(double lat1, double lon1, double lat2, double lon2) method. I cited its usage in the class method also.

* If you do re-use code from elsewhere (for example Stack Overflow), please cite the usage.

This rule is understood.

* The code you write should be of production quality, including relevant documentation to explain your solution, and most importantly, it should be code of which you are proud.

This rule is understood.

* Be creative, have fun with your solution, and ask questions!

I did email my initial README.md to my recruiter for team review on 11/18/2024.

**Bonus Questions**

* How would your program scale with many thousands of users? Can you modify your design for other key system considerations?

Currently, my application is a prototype, utilizing the H2 database for rapid development rather than an industrial-strength solution like PostgreSQL. To enhance scalability and performance, I would implement the following improvements:

1. **Database Upgrade and Optimization:**  
   Transition from H2 to PostgreSQL, leveraging PostGIS for geospatial indexing. This would significantly optimize spatial queries, such as identifying nearby user locations or national parks.
2. **Caching Layer:**  
   Introduce Redis as a caching layer for frequently accessed but infrequently updated data, such as static national park locations. This would reduce database load and improve response times.
3. **Real-Time Notifications:**  
   Implement WebSockets or Server-Sent Events (SSE) to enable real-time updates. This would allow the application to push notifications directly to users when the target user enters their designated radius.
4. **Containerization and Orchestration:**  
   Utilize Docker for containerization and Kubernetes for orchestration. This setup would allow for efficient scaling by deploying Kubernetes pods on a cloud platform, enabling automatic scaling up or down based on user demand.

* How would you determine bottlenecks?

To identify bottlenecks effectively, I would follow these steps:

1. **Set Goals:**  
   Define clear performance and scalability objectives, such as acceptable response times, throughput, or resource usage under various load conditions.
2. **Establish Benchmarks:**  
   Use realistic test scenarios to measure current performance and set baseline metrics for key operations.
3. **Profile the Application:**  
   Employ profiling and monitoring tools to analyze resource utilization and pinpoint bottlenecks. In past projects, I’ve leveraged tools like AppDynamics, JStack, and CA APM to:
   1. Identify slow-running database queries
   2. Detect cache misses and their impact on performance
   3. Resolve thread deadlocks and other concurrency issues.
   4. Evaluate and optimize thread pool sizes
4. **Iterative Optimization:**  
   Make changes incrementally, testing each modification to assess its impact on performance. This ensures that improvements are effective and do not introduce new issues.

* How might you handle additional mobile platforms with disparate API's?

To handle additional mobile platforms with disparate APIs, I would:

1. **Develop Common Interfaces:**

Develop abstract platform specific logic using interfaces and adapters that adhere to SOLID principles.

1. **Implement Platform Specific Adapters:**

Implement platform specific authentication and session management strategies.

1. **Use Platform Specific Data Models:**

Use data model conversion to ensure compatibility across platforms.

1. **Implement Logging and Monitoring:**

Implement centralized logging and monitoring for debugging and querying log files.

1. **Use API versioning:**

Use API versioning to manage backward compatibility of software releases.

1. **Test and Validate:**

Thoroughly test and validate data using appropriate tools for different platforms.

**Please reach out with any questions or difficulties.**