# Nearby Users

**Implemented Approach:**

Store the user information with latitude/longitude and zipcode details.

Use Haversine formula in SQL query to fetch the nearby users providing the current lat/long & radius

**Possible Alternatives:**

1. Move the haversine algorithm to the business layer of the API – for higher loads we can scale the API easily behind a load balancer.
2. Use secondary indexes on the User table for lat/long.
3. Use an algorithm to create a bounding box with max lat/long & min lat/long. This will reduce the dataset on which the query executes with the Haversine formula.
4. Use Geo-partitioning technique to partition the User data based on location information.
5. Create a shard based on the zipcode. Create an algorithm to get the list of zipcodes which fall within the radius from the current lat/long.
6. Use a GIS enabled datastore like PostGIS which supports querying nearby entities in their implementation.
7. To scale the API, we can containerize it and scale it behind a load balancer.

# Insert Bulk Users

**Implemented Approach:**

Use CSV to load the data into the User table directly. This is a synchronous implementation and will not scale for huge file sizes.

**Possible Alternatives:**

1. Implement an asynchronous upload process. Get the request (either as a file or direct payload) and implement workers in the backend to handle separate jobs to process the data and send the status asynchronously.