**Basic Requirements of the STP Site Selection Module Development:**

**Application Requirements**

* **Purpose**: Users have to select the optimum site for the sewage treatment plant.
* **Use Cases**: User will input the spatial data (particularly in the vector format) and will select the required conditioning factor data (could be in the raster / vector format) based on already uploaded data. Based on this input data and selected conditioning factor data, user have to select the optimum site for the sewage treatment plant.

**2. Data Information**

* **Data Format**: Shape files (.shp) and rasters (.tif)
* **Size and Scale**: Each data < 200 MB, overall data < 10 GB
* **Source**: User will upload only Shape files (.shp) if required, rest of the data will be stored in the backend. Additionally locational data will be accessed from the Google Map API.

**3. Technology Preferences**

* **Frontend**: Leaflet, Mapbox, or any other etc. whichever will perfect
* **Backend**: Node.js, Django, Flask or any other etc. whichever will perfect.
* **Database**: PostGIS, MongoDB with geospatial indexing or any other etc. whichever will perfect.
* **Hosting**: Any hosting preferences (AWS, Azure, self-hosted).

**4. Functional Details**

* **GIS Analysis**: Clipping, Weighted Overlay, Raster Calculator, Buffer Analysis, Euclidian Distance, Tabulate Area
* **Features**: Layer toggling, search functionality, heatmaps, etc.
* **Interactivity**: User inputs, drawing on maps, exporting data.

**5. APIs and Standards**

* **External APIs**: Google Maps, OpenStreetMap, ArcGIS map SDK for Java Script, or any other.
* **OGC Standards**: WMS, WFS, or other specific standards you need to implement.

**6. Performance and Scalability**

* **Users**: State Level Usage.

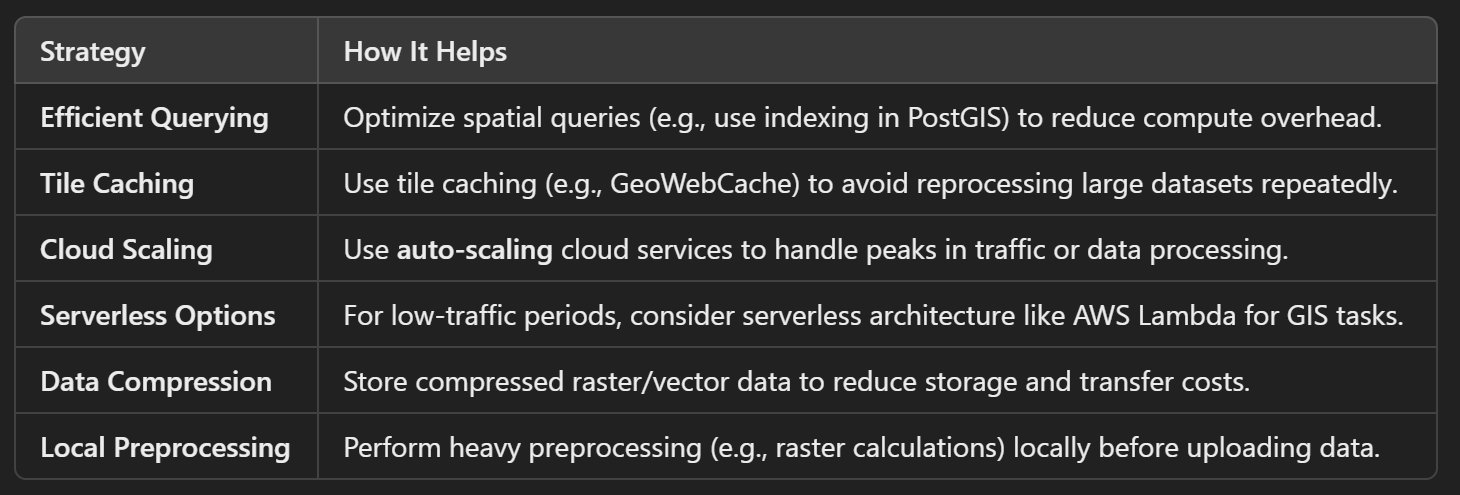
A screenshot of a computer

Description automatically generated

**When costs involved in this framework can escalate:**

1. **High User Traffic**:
   * A state-level usage scenario could involve hundreds or thousands of concurrent users performing GIS operations.
   * ***Solutions***: Load balancing, caching, and scaling servers dynamically.
2. **Large Datasets**:
   * Processing raster datasets (>GBs) for weighted overlays or raster calculations can strain compute resources.
   * ***Solutions***: Use efficient libraries like GDAL, batch processing, and cloud-based processing.
3. **Frequent Updates**:
   * If datasets are updated frequently (e.g., daily raster updates), there may be high data transfer or storage costs.
   * ***Solutions***: Automate processes with tools like Celery and optimize workflows.

**Strategy for optimizing the cost:**

****

**Complete Work Summary of WebGIS Development for the STP Site Suitability:**

**Phase 1: User Interaction (Frontend)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Component** | **Role/Contribution** | **Cost Overview** |
|  | Frontend (Leaflet) | * Provides an interactive map interface to users. * Users upload shapefiles, select layers, and trigger spatial analysis. | Free (Leaflet is open-source). Customization and hosting costs depend on usage. |
|  | Frontend + API Requests | * Communicates with the Django backend via API calls. * Sends user inputs (e.g., selected layers, analysis type) to the backend. | Minimal cost for API integration. Hosting costs depend on server traffic. |

**Phase 2: Data Processing (Backend)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Component** | **Role/Contribution** | **Cost Overview** |
|  | Django Backend | * Validates user-uploaded spatial data (e.g., shapefiles, GeoJSON). * Ensures correct file formats and CRS (Coordinate Reference System). | Free (Django is open-source). Hosting depends on server size and processing needs. |
|  | Django + PostGIS | * Preset locational data will be stored in PostGIS as spatial data. | Free (PostGIS is open-source). Hosting costs depend on database size and queries. |
|  | GeoDjango + PostGIS | * Stores spatial data in the PostGIS database. * Uses spatial models (PointField, PolygonField) to link the backend with PostGIS. | Free (GeoDjango is part of Django). Database hosting and scaling costs apply. |
|  | GeoDjango ORM | * Simplifies spatial operations like:   + Buffering   + Intersections   + Distance calculations | Free. Computational costs depend on query complexity and server resources. |
|  | GDAL (Optional) | * Preprocesses raster data (e.g., reprojection, cropping). * Supports raster calculations for weighted overlay or other analysis. | Free (GDAL is open-source). Costs may arise from processing large raster datasets. |
|  | Redis + GeoWebCache | * Caches frequently accessed raster/vector results for faster delivery. * Reduces redundant GDAL operations. | Redis has hosting costs depending on memory usage; GeoWebCache is open-source. |
|  | Celery + Redis | * Runs heavy GDAL-based tasks (e.g., raster clipping, suitability analysis) in the background. * Prevents frontend blocking and improves user experience. | Redis hosting and Celery integration costs (minimal if using free tools). |
|  | Google Maps Distance Matrix API | * Optionally calculates road-based proximity or travel time if precise routing is needed. | Pay-as-you-go model; pricing depends on API usage volume. |
|  | Django REST API | * Serializes processed spatial data into formats like GeoJSON. * Sends the nearest location details (coordinates, distance) to the frontend for display. * Sends results to the frontend for visualization. | Free. Hosting costs apply based on API traffic. |

**Phase 3: Data Publishing (GeoServer Integration)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Component** | **Role/Contribution** | **Cost Overview** |
|  | GeoServer | * Reads spatial data from PostGIS and publishes it as OGC-compliant services. | Free (GeoServer is open-source). Hosting costs depend on server usage. |
|  | GeoServer (WMS/WFS) | * Publishes raster data as WMS. * Publishes vector data as WFS. | Free (open-source). Hosting and scaling costs apply. |
|  | WMS/WFS URLs | * Provides WMS/WFS URLs that external GIS tools (e.g., ArcGIS, QGIS) can use for real-time data visualization. | No additional cost; data transfer costs depend on usage. |

**Phase 4: Data Access by External GIS Tools**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Component** | **Role/Contribution** | **Cost Overview** |
|  | ArcGIS/QGIS Clients | * Users load WMS/WFS URLs in ArcGIS or QGIS to access live spatial data. * GIS tools render spatial data in real-time for further analysis. | No cost for QGIS; ArcGIS requires a paid license. |
|  | Live Data Access | * GeoServer fetches live data from PostGIS and streams it to GIS tools via WMS/WFS. | Data transfer costs depend on traffic volume and hosting configuration. |

**Phase 5: Data Storage and Scalability**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Component** | **Role/Contribution** | **Cost Overview** |
|  | AWS S3 Storage | * Stores large raster/vector datasets for scalability. * Ensures efficient and cost-effective storage of spatial data. | AWS S3 charges based on storage size and data transfer. |
|  | Django + S3 Integration | * Retrieves spatial data from S3 for backend processing or user downloads. * Ensures processed outputs are saved and accessible. | Minimal cost for integration; S3 usage costs apply for storage and retrieval. |