**Software Requirements Specification (SRS)**

**1. Overall Description**

***1.1 Product Overview***

**Implementation:**

**WebApp:**  The application will be built using the Django framework, chosen for its robustness and efficiency in handling web-based applications. Django will handle routing, rendering templates, and managing interactions with both Excel and KMZ/KML files. WebApp will include interface that has following:

* **Interface:** Dashboard where user comes up with a lot/matricule number or pin address or radius and searches on the map to get information of the properties. Where property info is actually the Salesforce’s data which was directly available in the form of excel files + KML files for the properties on the map will be used to link the date from excel with the KML files
* **Customized Maps:** This includes embedding the map in the application and adding custom layers or markers as well as radius searching functionality.
* **Logic behind Efficiency:** We will use regional (not whole earth) customizable maps via any third-party provider like Maptive or any other tool to increase memory efficiency and scalability will be ensured to add other regions and their data to our app when needed.
* **ML:** Machine learning model data will also integrated in the property data shown in the maps.

**Maintenance:**

* **Framework Updates:** Regularly update Django and related libraries to the latest stable versions to ensure security and compatibility.

***1.2 Product Functions***

**Implementation:**

* **Data Integration:**
* **Data Synchronization:** Implement background jobs using Django's Celery to regularly sync properties between updated and the previous data.
* **Database Models:** Create Django models that represent the property data. These models will be used to store and manage data extracted from Salesforce excel and the data predicted by ML model.
* **Data Visualization:**
* **Map Integration:** Use KML files to render property data on the custom map. This includes adding markers, polygons, or custom layers to represent property details.
* **UI Enhancements:** Implement user interface components to allow users to interact with and view property data. Utilize libraries like Leaflet or D3.js for advanced visualizations if needed.
* **Performance Optimization:**
* **Load Testing:** Regularly conduct performance testing to ensure the application can handle peak loads. Optimize code and queries based on test results. Also the “Regional Maps” approach will help reducing memory load and optimizing user experience.
* **Code Optimization:** Optimize backend code and queries to reduce processing time. Use efficient algorithms and data structures.
* **Caching:** Implement caching strategies to minimize database hits and speed up data retrieval. Use Django’s caching framework or external caching services like Redis or Memcached.
* **Frontend Optimization:** Minimize and bundle JavaScript and CSS files. Implement lazy loading for map and data components.
* **Filtering Search Tool:**
* **Search Functionality:** Implement a search engine with Django’s query capabilities. Allow users to filter properties by various criteria (e.g., location, type and other fields available in the data).
* **Data Export:** Use libraries like pandas or openpyxl to export filtered data to Excel.
* **AI Features:**
* **AI Model Integration:** Develop machine learning models using libraries like TensorFlow or PyTorch to predict height and quality (blight). Deploy these models using Django’s model handling or a separate microservice.
* **Training and Evaluation:** Regularly update AI models with new data to improve accuracy. Monitor model performance and retrain as needed.

**Maintenance:**

* **Response Time Monitoring:** Use performance monitoring tools (e.g., New Relic, Datadog) to measure response times and ensure they meet the required performance standards.
* **Accuracy Evaluation:** Measure the accuracy of height calculations using a validation dataset. Aim for an accuracy rate of 95% for building height predictions.

***1.3 User Classes and Characteristics***

**Implementation:**

* **End Users:** Develop user roles and permissions in Django to provide different levels of access and functionality based on user type.
* **Administrators:** Create admin dashboards using Django’s admin interface to manage users, data, and application settings.

**Maintenance:**

* **User Management:** Regularly review user roles and permissions to ensure they are up-to-date and meet security requirements.
* **Admin Training:** Provide training and documentation for administrators to manage the application effectively.

***1.4 Operating Environment***

**Implementation:**

* **Server Environment:**
* **Deployment:** Use cloud-based high-performance servers (e.g., AWS EC2, Google Cloud Compute Engine) to deploy the application.
* **Scalability:** Implement auto-scaling and load balancing to handle high traffic and ensure reliable performance.
* **Client Environment:**
* **Compatibility Testing:** Regularly test the application on various browsers and devices to ensure cross-platform compatibility.
* **Framework:**
* **Django Configuration:** Configure Django settings for security, performance, and compatibility with the server environment.

**Maintenance:**

* **Server Monitoring:** Use monitoring tools (e.g., New Relic, Datadog) to track server performance and address issues promptly.
* **Browser Compatibility:** Keep track of updates to modern browsers and adjust the application as needed to ensure continued compatibility.

***1.5 Dependencies***

**Implementation:**

* **AI Libraries:** Select appropriate machine learning libraries and tools based on project needs and ensure they are well-supported.

**Maintenance:**

* **Library Updates:** Keep AI libraries and tools updated to leverage new features and maintain compatibility.

**2. External Interface Requirements**

***2.1 User Interfaces***

* **Web Interface:** The user interface (UI) will be designed using a combination of **HTML, CSS, and JavaScript**. It will feature interactive maps, data visualization panels, search tools, and AI-driven insights.
* **Implementation Details:**
* **Frontend Frameworks:** Use frontend libraries and frameworks such as **React** to ensure a modular and dynamic UI. These frameworks will help in rendering map data, handling user inputs, and managing UI components.
* **UI Components:** Integrate interactive UI components like dropdown menus, search bars, property data panels, and map tools using **JavaScript**. Utilize UI libraries such as **Bootstrap** or **Material-UI or tailwind**  to enhance the aesthetics and responsiveness of the web interface.
* **Map Interaction:** Users will interact with the map through actions like zooming, panning, clicking on properties, and searching for specific regions. These actions will trigger JavaScript event handlers to fetch data from the backend and update the UI.
* **Maintenance:**
* **UI/UX Updates:** Regularly update the UI based on user feedback and usability studies. Maintain a design system and update stylesheets and components to ensure consistent look and feel across devices.
* **Cross-browser Testing:** Conduct testing across multiple browsers (Chrome, Firefox, Edge, Safari) to ensure compatibility and responsiveness. Use tools like **BrowserStack** for comprehensive cross-browser testing.

**3. System Architecture**

***3.1 Overview***

The application will be structured using **Django's Model-View-Template (MVT) architecture**, ensuring a clean separation of concerns between data management (Model), user interface (Template), and control logic (View). **AI models** will be incorporated to handle building height calculations and visual quality assessments.

***3.2 System Components***

* **Frontend:**
* **HTML/CSS:** Used for structuring and styling the user interface to ensure a responsive, visually appealing design.
* **Implementation:** The application’s web pages will be created using Django’s templating system, which dynamically generates HTML from data provided by the backend. **CSS** will be used to style the pages, and modern CSS frameworks like **Bootstrap** will be employed to ensure a responsive design that works well on different devices (desktops, tablets, mobile).
* **Maintenance:** Regular updates to the HTML structure and CSS styles will be done based on feedback, testing, and evolving requirements. **Version control (e.g., Git)** will be used to manage changes, and **browser testing** tools will ensure compatibility across different devices and browsers.
* **JavaScript:** For interactive elements and handling client-side logic, such as map interactions, user input validation, and dynamic content loading.
* **Implementation:** JavaScript will be used to create interactive features like dynamic property information pop-ups on maps, filtering search tools, and real-time updates to the UI without requiring full page reloads. **AJAX** and **Fetch API** will be employed to communicate with the backend without reloading the page. Frontend libraries like **React** or **Vue.js** may be used for more complex, state-driven interactions.
* **Maintenance:** Client-side scripts will be optimized for performance, and modern JavaScript frameworks will be kept up to date. Testing frameworks like **Jest** or **Mocha** will be used to test JavaScript components.
* **Backend:**
* **Django:** This high-level Python framework will handle the server-side logic, and process data.
* **Implementation:** The Django application will be set up with distinct models for property data, user profiles, and system settings. These models will be linked to views that handle requests from the frontend (e.g., data retrieval, searches, etc.).
* **Maintenance:** The backend will be maintained by following the **Django ORM** best practices for handling models and database interactions. Automated tests will be implemented using Django’s built-in testing framework to ensure that the system remains stable during updates.
* **Database:** The application will store user data, property information, and application settings in a database like **PostgreSQL**.
* **Implementation:** The Django ORM will manage database interactions, handling the retrieval and updating of property data, user preferences, and AI results. **PostgreSQL** is a good fit due to its support for complex queries and spatial data (if needed for geographical mapping).
* **Maintenance:** Regular backups of the database will be scheduled, and migrations will be handled by Django’s migration system. Indexing will be used to optimize query performance.
* **AI Models:**
* **Machine Learning Libraries:** The AI models for building height calculations and quality assessments will be built using libraries such as **TensorFlow**, **PyTorch**, or **scikit-learn**.
* **Implementation:** The AI models will be trained offline using labeled datasets that include building images and height/quality annotations. Once trained, the models will be deployed within the Django framework, where they will process new building images submitted by the user or retrieved from Google Maps. The predictions will then be returned to the user interface for display.
* **Maintenance:** The AI models will require periodic retraining as new data becomes available or to improve accuracy. Tools like **TensorBoard** can be used to monitor model performance, and version control will be applied to keep track of model updates.
* **Data Integration:**
* **KMZ/KML and Excel files:** The excel files will be integrated in the maps by uploading the files in the form of KML directly into the webapp by the admin.

***3.3 Data Flow***

* **User Interaction:**
* **Frontend Implementation:** Users will interact with the web application through the frontend interface, selecting areas, clicking on properties, and using filtering tools. These interactions will trigger events captured by JavaScript, which will send requests to the Django backend.
* **Maintenance:** Frontend code will be kept efficient to avoid unnecessary data transfers. Continuous improvements based on user feedback will keep the interface intuitive and responsive.
* **Data Retrieval:**
* **Backend Implementation:** When a user selects an area or clicks on a property, the Django backend will send a request to the database to retrieve relevant property data. This data will then be processed and sent to the Webapp to display on the map.
* **AI Processing:**
* **Backend Implementation:** AI models integrated into the backend will analyze images of buildings, calculating heights or assessing visual quality. This data will be overlaid on the map, providing additional insights to the user.
* **Maintenance:** AI models will be retrained periodically, and their performance will be monitored. New datasets will be incorporated to improve the accuracy of predictions. Deployment pipelines will ensure that new models are tested before going live.
* **Search and Export:**
* **Frontend and Backend Implementation:** Users will use filtering tools to search for properties based on criteria like location, price, and building quality. The Django backend will query the database and return the filtered results. Users can export the search results to **Excel** for further analysis.
* **Maintenance:** The export functionality will be regularly tested to ensure that all data is correctly formatted and no issues arise with different versions of Excel. New filtering options can be added over time as user43r needs evolve.

**4. Non-Functional Requirements**

***4.1 Security***

* **Data Protection:**
* **Implementation:**
* All sensitive data (such as user information and property details) will be encrypted both at rest and in transit. **SSL/TLS** certificates will be used to secure data transmission. Sensitive user data (e.g., passwords) will be hashed using algorithms like **bcrypt**.
* **Maintenance:**
* Security audits will be performed regularly to identify and fix vulnerabilities. Tools like **OWASP ZAP** will be used to test for potential security threats. The application’s dependencies will be regularly updated to mitigate security risks.
* **User Authentication:**
* **Implementation:**
* A secure login and authentication system will be implemented using **Django’s built-in authentication** and **third-party libraries** like **django-allauth** for social logins or **OAuth2**. User roles will be enforced via **access control mechanisms** to restrict certain functionalities to administrators.
* **Two-factor authentication (2FA)** can be introduced for additional security for administrators.
* **Maintenance:**
* Security patches will be applied promptly, and user authentication logs will be monitored for suspicious activities. Password policies will be enforced to ensure strong user credentials.

***4.2 Usability***

* **Ease of Use:**
* **Implementation:**
* The application will be designed with an intuitive user interface, ensuring that users can navigate, search, and interact with the system easily. **User experience (UX) best practices** will be applied to design clean, clear, and responsive interfaces.
* Features like **autocomplete**, **drag-and-drop functionality**, and clear error messages will improve user interactions. **Accessibility standards (WCAG)** will be followed to ensure the app is usable for a diverse audience.
* **Maintenance:**
* Regular usability testing will be conducted, and feedback loops will be established to gather input from end-users. The user interface will be continuously improved based on insights from user testing and performance metrics.
* **Documentation:**
* **Implementation:**
* Comprehensive **user guides** and **online help resources** will be provided for end users and administrators. These will include step-by-step tutorials, FAQs, and video guides for complex features.
* A dedicated **documentation site** will be developed using tools like **Sphinx** or **MkDocs**, and kept updated as new features are added.
* **Maintenance:**
* Documentation will be versioned to match the application’s development cycle. Updates will be made whenever significant changes are implemented, ensuring that users always have access to current instructions.

**5. Other Requirements**

***5.1 Legal and Regulatory Requirements***

* **Data Privacy:**
* **Implementation:**
* The application will adhere to data protection regulations, such as the **General Data Protection Regulation (GDPR)**. This will include implementing features like:
* **User Consent:** Ensure that users provide explicit consent for data collection and usage, particularly for personal data such as contact details, property information, etc.
* **Data Anonymization:** Personal information will be anonymized where necessary, reducing the risk of data exposure.
* **Right to Access/Erasure:** Provide users with the ability to request access to their stored data and the option to request its deletion.
* **Data Breach Notifications:** A system will be implemented to detect data breaches and notify affected users within 72 hours, as per GDPR requirements.
* **Encryption:** All user data will be encrypted in transit and at rest using industry-standard protocols like **SSL/TLS** and **AES-256** to ensure data security.
* **Maintenance:**
* **Legal Audits:** Regular audits will be conducted to ensure continued compliance with regulations. Legal professionals will be consulted to keep up-to-date with changes in regulations.
* **Privacy Policy Updates:** The application's privacy policy will be regularly reviewed and updated to reflect current practices and regulations.
* **User Requests:** Processes will be maintained to handle user requests related to data access, portability, or deletion within a legally mandated timeframe.
* **Accessibility:**
* **Implementation:**
* The application will comply with **Web Content Accessibility Guidelines (WCAG 2.1)** to ensure it is accessible to users with disabilities. This includes:
* **Keyboard Navigation:** Ensuring that all interactive elements are accessible via keyboard navigation alone.
* **Screen Reader Compatibility:** Implementing ARIA (Accessible Rich Internet Applications) labels and roles to ensure that content is properly conveyed to users relying on screen readers.
* **Contrast Ratios:** Ensuring a high contrast ratio for text and graphical elements to accommodate users with visual impairments.
* **Alt Text:** Providing descriptive alternative text for images and non-text elements for accessibility.
* **Maintenance:**
* **Accessibility Testing:** Regular testing using accessibility evaluation tools like **WAVE** or **Axe** will be conducted to ensure continued adherence to WCAG standards.
* **User Feedback:** Feedback will be collected from users with disabilities to identify areas of improvement in accessibility. Accessibility experts will be consulted for continuous enhancements.
* **Updates for Compliance:** The development team will stay informed about changes in accessibility standards and update the application accordingly.

***5.2 Environmental Requirements***

* **Hosting:**
* **Implementation:**
* The application will be hosted on a reliable and scalable cloud service provider, such as **AWS**, **Google Cloud Platform**, or **Microsoft Azure**. These platforms offer:
* **Auto-Scaling:** Automatically adjusting resources to accommodate changes in traffic, ensuring consistent performance during high-demand periods.
* **Sustainability Initiatives:** Many cloud platforms have environmental initiatives aimed at minimizing the carbon footprint of data centers. Deploying on such platforms can help meet environmental sustainability goals.

**6. Testing and Deployment**

***6.1 Testing***

**1. Functional Testing**

* **Unit Testing:**
* **Tool:** pytest, unittest
* **Purpose:** Validate individual components or functions of the application.
* **System Testing:**
* **Tool:** Selenium, Postman
* **Purpose:** Validate the complete and integrated application to ensure it meets the specified requirements.

**2. Performance Testing**

* **Load Testing:**
* **Tool:** Apache JMeter, Gatling
* **Purpose:** Assess the application's performance under various load conditions to ensure it handles traffic efficiently.
* **Stress Testing:**
* **Tool:** Locust, BlazeMeter
* **Purpose:** Determine the application's breaking point and how it recovers from failures.

**3. Security Testing**

* **Vulnerability Scanning:**
* **Tool:** OWASP ZAP, Nessus
* **Purpose:** Identify security vulnerabilities and potential threats in the application.
* **Penetration Testing:**
* **Tool:** Burp Suite, Kali Linux
* **Purpose:** Simulate attacks to identify security weaknesses and assess the application's defenses.

**4. Usability Testing**

* **Tool:** UsabilityHub, Crazy Egg
* **Purpose:** Evaluate the user interface and user experience to ensure the application is intuitive and user-friendly.

**5. AI Model Testing**

* **Tool:** TensorFlow, PyTorch
* **Purpose:** Validate the accuracy and performance of AI models used for building height calculation and quality assessment.

***6.2 Deployment***

**1. Continuous Integration/Continuous Deployment (CI/CD)**

* **Tool:** Jenkins, GitHub Actions, GitLab CI
* **Purpose:** Automate the build, testing, and deployment processes to ensure code changes are integrated smoothly and consistently.

**2. Deployment Automation**

* **Tool:** Docker, Kubernetes
* **Purpose:** Containerize the application for consistent deployment across different environments and manage the deployment using orchestration tools.

**3. Web Hosting**

* **Tool:** AWS, Azure, Google Cloud Platform
* **Purpose:** Host the web application on scalable and reliable cloud platforms to ensure high availability and performance.

**4. Database Management**

* **Tool:** pgAdmin, Django ORM
* **Purpose:** Manage and monitor database performance and ensure data integrity.

**5. Monitoring and Logging**

* **Tool:** Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana)
* **Purpose:** Monitor application performance, collect logs, and analyze system health to detect and troubleshoot issues.

**6. Backup and Recovery**

* **Tool:** AWS Backup, Azure Backup
* **Purpose:** Regularly back up application data and configurations to ensure data recovery in case of failures.

**7. User Acceptance Testing (UAT)**

* **Tool:** Manual testing based on user scenarios
* **Purpose:** Ensure the application meets the end users’ needs and requirements before the final deployment.

**7. Project Timeline (6 Months – 9000USD)**

(5% Advance ~ 500USD)

**NOTE**:

-We agree that all code be given weekly for your programmer to verify our work (Available on GitHub)

-We also agree that our workers can be tracked with Hubstaff

**1. Design Phase (Weeks 1-2) (2 weeks)**

**Key Deliverables:**

* **System Architecture Design**: Finalize the architecture, including database schema, API structure, and module interactions.
* **UI/UX Design**: Create wireframes and prototypes for key user interfaces.
* **Design Review**: Get approval from stakeholders on the system architecture and UI/UX designs.

**Milestone**:

In this milestone we will be providing you a Figma/xd design for the web app (that you agree with).

**(700USD)**

**2. Frontend Development + Some Backend Development + Starting training A.I models (Weeks 3-12) (10 weeks)**

**Key Deliverables:**

* **UI Development**: Develop the frontend based on the approved designs using React, Bootstrap, etc.
* **Map Integration**: Implement the mapping tools and ensure they communicate effectively with the backend.
* **Properties Integration**: Integrating KML to maps so that they can be viewed properly + Integrating property data (Excel data gotten from salesforce)
* **User Authentication and Security**: Implement and test user authentication, authorization, and other security measures.
* **Database Setup**: Implement the database schema and configure connections.
* **API Development**: Start Development and testing of APIs for data integration, synchronization, and other backend services.
* **Upload Functionality**: This would allow users to upload property data specific to different regions of the country. This data could include property listings, geographical coordinates, and other relevant attributes. When new data is uploaded, the system will process it and display the corresponding points on a map in real-time or upon fresh.

**Milestone (What we will be able to do with the app):**

* Complete the frontend implementation of the web app UI from Milestone 1.
* user authentication
* Users will be able to view detailed property information, including past sales data, by clicking on properties displayed on the map.
* a search function will be implemented, allowing users to search for properties by address or lot number.
* The training of AI models will also commence during this phase.
* Upload new areas to the map (KML files + property data)

**(3000USD)**

**3. Backend Development and Integration ( Weeks 13-18) (6 weeks)**

**Key Deliverables:**

* **Database Setup**: Finalize Implementing the database schema and configure connections.
* **API Development**: Finalize Development and testing of APIs for data integration, synchronization, and other backend services.
* **Radius Search**: Functionality integration for a user to search an area in a radius to get its properties + export selected properties to excel
* **AI/ML Model Integration**: Finish integrating machine learning models into the backend.
* **Integration**: Integrating developed modules with the frontend.

**Milestone (What we will be able to do with the app):**

Backend development for the APIs created to serve the frontend in Milestone 2 will be completed.

You’ll be able to:

* use the radius search tool (Select multiple properties and view the data in the program + be able to export it to excel (past sales / property info / both)
* Use the A.I height tool (98-99 % + accurate)
* Use the A.I quality tool (98-99 % + accurate)

NOTE: If one or both of the A.I tool is not to your satisfaction, the price of this deliverable will be reduced to reflect that, and further testing will be done in the next phase. Once the tool is to your satisfaction we will charge for it.

**(2500USD)**

**4. Testing Phase (Weeks 19-22) (4 weeks)**

**Key Deliverables:**

* **Functional Testing**: Conduct comprehensive functional testing to ensure that the system meets all requirements.
* **Performance Testing**: Test the system’s performance under various loads and optimize where necessary.
* **Security Testing**: Conduct penetration testing to identify and fix security vulnerabilities.
* **User Acceptance Testing (UAT)**: Perform UAT with stakeholders to validate that the system meets their expectations.

**Milestone**:

All Testing Completed and UAT Approved. Proof of testing will be given.

**(1500USD)**

**5. Finalization and Deployment (Weeks 23-24) (2 weeks)**

**Key Deliverables:**

* **Documentation**: Complete all technical and user documentation, including deployment guides and user manuals.
* **Final Project Review**: Review the project with stakeholders, including a retrospective to discuss what went well and what could be improved.
* **Project Closure**: Officially close the project after ensuring all deliverables are met and all issues are resolved.

**Milestone**:

Give documentation + assure customer satisfaction

**(1300USD)**

**6. Post-Deployment Support (always available)**

**Key Deliverables:**

* **Monitoring and Bug Fixing**: Monitor the system for any post-launch issues and address them promptly.

**7. GRAND TOTAL + SOHYL:**

For Sohyl he assumes ~ 4 hours/ week for verifying work (some of the weeks the company will be testing the model, so theres no code to really verify, so I’m assuming 20 weeks of code to verify).

20 CAD \* 4 hours \* 20 weeks = 1600 CAD

9000 USD to cad = 12,195 CAD

12,195 + 1,600 = $13,795 CAD

**8. Revision Policy:**

The **Revision Policy** indicates that the project allows for unlimited revisions to the design and implementation as long as the original requirements remain unchanged. However, if the project requirements are modified, the revision policy may need to be reassessed. Here’s a deeper explanation:

* **Unlimited Revisions with Unchanged Requirements**:
* The project framework, which includes using Django for web application development, allows for flexibility and adaptability. Django’s modular architecture makes it easier to make changes to templates, views, and models without affecting the overall system. As long as the core requirements (like data handling, mapping, and machine learning integration) remain the same, the team can make as many revisions as needed to refine or optimize the implementation.
* This is crucial in an Agile development process, where iterative feedback and changes are a part of continuous improvement. Unlimited revisions ensure that the project evolves according to user feedback and technical enhancements without incurring additional costs or requiring renegotiation, as long as these changes do not alter the original scope of the project.
* **Reassessment Upon Requirement Changes**:
* If there is a change in requirements, such as altering the data integration method, introducing new AI models, these changes might affect the design and implementation significantly. This would require a reassessment of the revision policy because:
* **Scope Creep**: Changing requirements often lead to scope creep, which could necessitate additional resources, time, or budget.
* **Technical Repercussions**: For instance, a change in the map integration tool (from Maptive to another GIS solution) may require rewriting parts of the code, reconfiguring the server environment, or retraining AI models. This would extend the development timeline and potentially require new tools or libraries.
* **Reevaluation of Contracts and Agreements**: New requirements might require updating the contractual agreements, including timelines, deliverables, and costs.

**Floorplate A.I**

**Flooring Plating Area Calculation (~80% accurate)**

**Price: $1,200**

Calculate floor area from satellite images to estimate plating costs, using two approaches:

**Scenario 1:** Using Existing Data for Model Training

* **Data Collection:** Use previous known floor areas.
* **Model Development:** Train a CNN in PyTorch to predict floor area from images.
* **Integration:** Embed the trained model into the platform for users to upload areas and receive area predictions.
* **Testing:** Validate model performance and refine as needed.

**Scenario 2:** Edge Detection for Area Calculation

* **Edge Detection:** Use algorithms like Canny to detect building edges from satellite images.
* **Area Calculation:** Convert detected edges into real-world area using image scale.

**Implementation:** Allow users to upload images and receive area calculations directly.

**Timeline**: This take 2-3 weeks to complete

**Integration**: Integrating developed modules with the frontend.