Module Guide for Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

April 2, 2025

1 Revision History

Date	Version	Notes
January 17th	1.0	

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

See SRS Documentation here

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design for OrbitWatch follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Change in labeling tool(s) used or the introduction of additional labeling tools.

AC2: Change in form parameters for project creation and account registration workflows.

AC3: Change in the set and presentation of project statistics displayed to clients.

AC4: Changes in algorithms used to calculate image priors.

AC5: Changes in algorithms for object extraction from images.

AC6: Changes in algorithms that update and refine labeller expertise metrics.

AC7: Changes in image serving and distribution algorithms.

AC8: Change in satellite image providers or addition of multiple providers, requiring adjustments in data ingestion modules.

AC9: Change in backend hosting services (e.g., switching or integrating additional cloud providers beyond AWS/Azure) affecting deployment modules.

AC10: Change in consensus algorithms or user accuracy models used to resolve labeling conflicts, impacting the data validation modules.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The predefined user access levels and roles (e.g., admin, labeller, client) are considered stable.

UC2: The choice of database technology (e.g., relational database) and its core schema structure is fixed early on.

UC3: The foundational languages and frameworks selected for development (e.g., Node.js for backend services, Typescript for frontend) are assumed to remain constant.

UC4: Decisions around the overall architectural pattern (such as microservices) are made early and considered stable.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 3. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Account Creation Interface

M3: Account Database Connector

M4: Account Database

M5: Account Update Interface

M6: Login Interface

M7: Access Token

M8: Labeler

M9: Client

M10: User

M11: Account Creation Controller

M12: Account Update Controller

M13: Authentication Controller

M14: Satellite Image Request Interface

M15: Satellite Image Request Controller

M16: Satellite Image Request

M17: Project Creation Interface

M18: Project Creation Controller

M19: Tile Creation Interface

M20: Tile Creation Controller

M21: Project

M22: Service Request Failure Interface

M23: Image Upload Interface

M24: Report Interface

M25: Report Controller

M26: Report

M27: Project Selection Interface

M28: Project Selection Controller

M29: Labeling Interface

M30: Labeling Controller

M31: Image

M32: Label Server

M33: Label Database Connector

M34: Label Database

M35: ImageObject Database Connector

M36: ImageObject Database

M37: Labeller Database Connector

M38: Labeller Database

M39: Object Extraction Manager

M40: Image Service Manager

M41: ModelCreation (Abstract Class)

M42: CNNModelCreation

M43: OtherModelCreation

M44: ModelManager

M45: MLModelDatabase

M46: Label Confidence Service

M47: Object Extraction Service

M48: Image Prior Analyzer

M49: Labeller Expertise Calculator

M50: Image Mask Service

M51: Image Selection Service

M52: ModelComparision Evaluation

M53: CrossValidation Evaluation

M54: ModelTrainingService

M55: ModelEvaluationService

M56: Project Manager

M57: Project Collection Manager

M58: Project Database Connector

M59: Core Image Database Connector

M60: Planet Labs Connector

M61: Report Manager

Level 1	Level 2
Hardware-Hiding Module	

Table 1: Module Hierarchy

Level 1	Level 2
	Account Creation Interface
	Account Database
	Account Update Interface
	Login Interface
Behaviour-Hiding	Access Token
Module	Labeler
	Client
	User
	Satellite Image Request Interface
	Satellite Image Request
	Project Creation Interface
	Project
	Service Request Failure Interface
	Image Upload Interface
	Report Interface
	Report
	Project Selection Interface
	Labeling Interface
	Image
	Label Server
	Label Database Connector
	Label Database
	ImageObject Database Connector
	ImageObject Database
	Labeller Database Connector
	Labeller Database
	Object Extraction Manager
	Image Service Manager
	ModelCreation (Abstract Class)
	CNNModelCreation
	OtherModelCreation
	ModelManager
	MLModelDatabase

Table 2: Module Hierarchy

.

Level 1	Level 2
	Account Creation Controller
	Account Database Connector
	Account Update Controller
	Authentication Controller
Software Decision	Satellite Image Request Controller
Module	Project Creation Controller
	Report Controller
	Project Selection Controller
	Labeling Controller
	Label Confidence Service
	Object Extraction Service
	Image Prior Analyzer
	Labeller Expertise Calculator
	Image Mask Service
	Image Selection Service
	ModelComparision Evaluation
	CrossValidation Evaluation
	ModelTrainingService
	ModelEvaluationService
	Tile Creation Interface
	Tile Creation Controller

Table 3: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 4.

There are several key high level design decisions that were made to ensure the requirements outlined in the SRS are met. First, the system has been decomposed into several microservices. These services all connect to the same databases, but have the flexibility to operate on their own cadence. Each service can be mirrored by a duplicate service to improve capacity. These decisions are critical to ensuring the speed, uptime, capacity, and latency requirements. Each of the key algorithms in the system have been encapsulated in their own module. These include the object extraction service, image serving, ML model creation, and report generation, among others. This gives the development team the flexibility to experiment with alternative algorithms, allowing continuous improvement towards performance and accuracy requirements. The authentication service is designed to meet the authentication requirements by ensuring only approved users can access privileged information. Every functional requirement is covered by the functionality of a subsystem.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Account Creation Interface (M2)

Secrets: The design format of the account creation user interface.

Services: Displays a form to collect user information and submits that information to be processed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.2 Account Database (M4)

Secrets: The data structure, storage, and access mechanisms for account related data.

Services: Can store, insert, update and retrieve user account data.

Implemented By: OrbitWatch

Type of Module: Abstract Data Type

7.2.3 Account Database Connector (M3)

Secrets: The database access key and database access algorithms.

Services: Can request storage, insertion, updates and retrieval of user account data from the database.

Implemented By: OrbitWatch

Type of Module: Abstract Data Type

7.2.4 Account Update Interface (M5)

Secrets: The design format of the update account information user interface.

Services: Displays a form with the users current information that can be modified and submits any updated information to be processed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.5 Login Interface (M6)

Secrets: The design format of the login user interface.

Services: Displays a form to collect login credentials and submits that information to be processed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.6 Access Token (M7)

Secrets: Token structure, encryption, expiration, and renewal mechanisms.

Services: Can determine if a user's token has expired and allows the token to be renewed.

Implemented By: OrbitWatch

Type of Module: Abstract Data Type

7.2.7 Labeler (M8)

Secrets: The format and structure of Labeler data.

Services: Takes in the necessary data to create a Labeler.

Implemented By: OrbitWatch

Type of Module: Record

7.2.8 Client (M9)

Secrets: The format and structure of Client data.

Services: Takes in the necessary data to create a Client.

Implemented By: OrbitWatch

Type of Module: Record

7.2.9 User (M10)

Secrets: The format and structure of User data.

Services: Takes in the necessary data to create a User.

Implemented By: OrbitWatch

Type of Module: Record

7.2.10 Satellite Image Request Interface (M14)

Secrets: The design format of the interface for requesting satellite images.

Services: Displays a form to collect specifics on the satellite images, calculates estimated cost based on current form entries and submits the form information to be processed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.11 Satellite Image Request (M16)

Secrets: The format and structure of the data needed for a satellite image request.

Services: Takes in the necessary data to create a satellite image request.

Implemented By: OrbitWatch

Type of Module: Record

7.2.12 Project Creation Interface (M17)

Secrets: The design format of the project creation interface.

Services: Displays a form to collect project details, calculates an estimated cost using these details and submits that information to be processed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.13 Project (M21)

Secrets: The format and structure of the data needed for a Project.

Services: Takes in the necessary data to create a Project.

Implemented By: OrbitWatch

Type of Module: Record

7.2.14 Service Request Failure Interface (M22)

Secrets: The design format of the request failure interface.

Services: Displays a warning to users that something went wrong.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.15 Image Upload Interface (M23)

Secrets: The design format of the image upload interface.

Services: Allows users to upload images from their device, and validates they are images of the correct format.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.16 Report Interface (M24)

Secrets: The design format of the summary report interface.

Services: Displays statistics and results of a specific project.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.17 Report (M26)

Secrets: The format and structure of Report data.

Services: Takes in the necessary data to create a Report.

Implemented By: OrbitWatch

Type of Module: Record

7.2.18 Project Selection Interface (M27)

Secrets: The design format of the project selection interface.

Services: Display all available projects that a user can label images from.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.19 Labeling Interface (M29)

Secrets: The design format of the labeling interface.

Services: Displays an image to be labeled, displays label choices which can be selected, and allows images to be skipped.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.20 Image (M31)

Secrets: The format and structure of Image data.

Services: Takes in the necessary data to create a Image.

Implemented By: OrbitWatch

Type of Module: Record

7.2.21 Label Server (32)

Secrets: The steps to accept and store a label.

Services: Converts a label JSON object and passes it to be stored in the database.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.22 Label Database Connector (33)

Secrets: The steps to connect to and manipulate a database.

Services: Connects to a database and executes pushing and fetching of data.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.23 Label Database (34)

Secrets: The storing and retrieving of data.

Services: Allows connection by a database connector to facilitate the storage and retreival of data.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.24 ImageObject Database Connector (35)

Secrets: The steps to connect to and manipulate a database.

Services: Connects to a database and executes pushing and fetching of data.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.25 ImageObject Database (36)

Secrets: The storing and retrieving of data.

Services: Allows connection by a database connector to facilitate the storage and retreival of data.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.26 Labeller Database Connector (37)

Secrets: The steps to connect to and manipulate a database.

Services: Connects to a database and executes pushing and fetching of data.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.27 Labeller Database (38)

Secrets: The storing and retrieving of data.

Services: Allows connection by a database connector to facilitate the storage and retreival of data.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.28 Object Extraction Manager (39)

Secrets: The algorithms used to aggregate labels into usable ImageObjects for Model training.

Services: Calls other modules to efficiently compute the most likely set of objects from a set of independent labels.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.29 Image Service Manager (40)

Secrets: The algorithms used to select images to be served to a user.

Services: Calls other modules to select the next images to be server to a user.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.30 ModelCreation (Abstract Class) (M41)

Secrets: The abstract details of how machine learning (ML) model creation is organized, including any hidden complexities of managing configuration parameters.

Services: Defines the interface for creating ML models. Specific implementations (e.g., Convolutional Neural Networks, Support Vector Machines) extend this class to instantiate the appropriate model structure.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.31 CNNModelCreation (M42)

Secrets: The underlying architecture for a convolutional neural network model, including details about layer definitions, activation functions, and other internal configurations.

Services: Accepts layer configurations and constructs a CNN model suitable for training or inference tasks.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.32 OtherModelCreation (M43)

Secrets: The hyperparameter configurations and implementation details necessary for creating alternative model types (e.g., Decision Trees, SVMs).

Services: Allows the instantiation of various ML models by applying specific parameters (such as maximum tree depth or kernel functions) to build a model instance.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.33 ModelManager (M44)

Secrets: The internal logic that governs model lifecycle management (e.g., statuses such as "Training," "Evaluating," or "Completed") as well as the way creation and update timestamps are maintained.

Services: Manages ML models by creating new instances, updating their status, retrieving existing models, and deleting models as needed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.2.34 MLModelDatabase (M45)

Secrets: The database schema, connection details, and query mechanisms for storing and retrieving ML model metadata and configurations.

Services: Provides persistent storage for ML model data, supporting operations such as create, read, update, and delete (CRUD).

Implemented By: OrbitWatch

Type of Module: Abstract Data Type

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Account Creation Controller (M11)

Secrets: Form validation and account creation algorithms.

Services: Validates the form information, creates an account with that information, and can upload the account to the database.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.2 Account Update Controller (M12)

Secrets: Form validation and account modification algorithms.

Services: Gets current user information, validates the update form information and can pass account updates to the database.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.3 Authentication Controller (M13)

Secrets: Credential validation, access token validation, and access token generation algorithms.

Services: Validates credentials given by user, generates access tokens, and validates access tokens.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.4 Satellite Image Request Controller (M15)

Secrets: Form validation and image request algorithms.

Services: Validates the form information, creates and sends a request to a third party for the necessary pictures.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.5 Project Creation Controller (M18)

Secrets: Form validation and project creation algorithms.

Services: Validates project information provided and creates a new project.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.6 Tile Creation Interface (M19)

Secrets: Tile creation algorithms and related tools.

Services: Provides a user interface for creating tiles.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.7 Tile Creation Controller (M20)

Secrets: Tile creation algorithms and related tools.

Services: Validates incoming images and creates a new tiles in the database.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.8 Report Controller (M25)

Secrets: Logic for getting statistics of a project and exporting images onto the users device.

Services: Get statistics and labeled images of project, and exports images to external devices.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.9 Project Selection Controller (M28)

Secrets: Logic for getting active projects and redirecting labelers upon selection.

Services: Gets valid projects that are currently available, and redirects labelers to a selected project labeling interface.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.10 Labeling Controller (M30)

Secrets: Algorithms for label creation, removal and submission.

Services: Creates label for an image, removes label from an image, and submits labels to be processed.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.11 Label Confidence Service (46)

Secrets: The algorithms used to determine confidence in detected ImageObjects.

Services: Calculates the confidence of a detected ImageObject.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.12 Object Extraction Service (47)

Secrets: The algorithms used to extract ImageObjects.

Services: Calculates the most likely ImageObjects from a set of labels and priors.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.13 Image Prior Analyzer (48)

Secrets: The algorithms used to calculate ImagePriors.

Services: Calculates the prior liklihood of a pixel being relevant.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.14 Labeller Expertise Calculator (49)

Secrets: The algorithms used to calculate Labeller Expertise.

Services: Calculates the expertise of a labeller in a given class using previous labels.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.15 Image Mask Service (50)

Secrets: The algorithms used to modify a given image.

Services: Modifies an image in an attempt to improve labelling accuracy or efficiency.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.16 Image Selection Service (51)

Secrets: The algorithms used to select the next image to show a user.

Services: Selects an image to show to a given user.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.17 ModelComparision Evaluation (M52)

Secrets: The evaluation metrics and internal comparison algorithms (e.g., accuracy, F1-score) used to contrast multiple ML models.

Services: Evaluates one or more models against provided test data, calculates a set of performance metrics, and produces a comparison summary or report.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.18 CrossValidation Evaluation (M53)

Secrets: The logic behind performing K-fold cross-validation (or alternative validation schemes), including how folds are formed and results are averaged.

Services: Executes cross-validation on a given model and dataset, aggregates the results of each fold, and computes final performance metrics such as mean accuracy, mean precision, etc.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.19 ModelTrainingService (M54)

Secrets: The training algorithms, hyperparameter optimization logic, and internal routines for monitoring the status of model training.

Services: Orchestrates the process of training ML models using specified data and configuration parameters; can also halt training if requested.

Implemented By: OrbitWatch

Type of Module: Abstract Object

7.3.20 ModelEvaluationService (M55)

Secrets: The design and algorithms for evaluating models, combining metrics, and aggregating results for final output.

Services: Evaluates a trained model on test data, computes relevant performance metrics, and exposes them for reporting or further analysis.

Implemented By: OrbitWatch

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Table 4: Trace Between Requirements and Modules

Req.	Modules
FR0	M11, M2, M3, M4, M10, M9
FR1	M13, M6, M7, M3, M4, M10, M9
FR2	M13, M7, M3, M4, M12, M5, M10, M9
FR4	M17, M18, M56, M19, M20
FR5	M24, M25, M61, M26
FR6	M23, M56
FR7	M14, M15, M16, M56
FR8	M22, M56, M19, M20
FR9	M11, M2, M3, M4, M10, M9
FR10	M13, M6, M7, M3, M4, M10, M8
FR11	M13, M7, M3, M4, M12, M5, M10, M9
FR13	M29, M30, M32, M34, M33, M40, M51
FR14	M24, M25, M39, M46, M47, M48, M50
LF0	M5, M6, M2, M14, M22, M17, M23, M24, M27, M29, M19, M20
LF1	M5, M6, M2, M14, M22, M17, M23, M24, M27, M29, M19, M20
LF2	M5, M6, M2, M14, M22, M17, M23, M24, M27, M29, M19, M20
UH0	M5, M6, M2, M14, M22, M17, M23, M24, M27, M29, M19, M20

Continued on next page

Req.	Modules
UH1	M5, M6, M2, M14, M22, M17, M23, M24, M27, M29, M19, M20
UH2	M5, M12
UH6	M29
UH7	M22
UH9	M29
PR0	M2, M11, M9
PR1	M2, M11, M8
PR2	M24, M25, M26, M61
PR3	M29, M30, M31, M40, M51
PR5	M24, M25, M39, M46, M47, M48, M50
PR6	Related to all modules
PR7	Related to all modules
PR8	M23, M14, M15, M16, M50, M40, M51, M48, M36, M35
PR9	Related to all modules
OE0	Related to all modules
OE1	M14, M15, M16
OE4	M41, M42, M43, M54, M55
OE5	M36, M35, M54, M55, M44, M41
MR0	Related to all modules
SE0	M13, M7, M29, M30, M27, M28
SE1	M13, M7, M17, M18, M19, M20
SE2	M2, M11
SE3	M2, M11
SE4	M22
SE5	M11, M12, M18, M39, M46, M47, M48, M50
SE6	M56, M30
SE7	M4, M3, M36, M35, M37, M38, M33, M34, M45
SE8	M12, M11, M4
SE10	M3, M35, M37, M33

AC	Modules
AC1	M29
AC2	M17, M2
AC3	M24, M26
AC4	M48
AC5	M47
AC6	M49
AC7	M40
AC8	M15
AC10	M46

Table 5: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 12 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

Level 1: Foundational Modules

- **Label Database**
  - **Secrets:** The storing and retrieving of data.
  - **Services:** Allows connection by a database connector to facilitate the storage and retrieval of data.
- **ImageObject Database**
  - **Secrets:** The storing and retrieving of data.
  - **Services:** Allows connection by a database connector to facilitate the storage and retrieval of data.
- **Labeller Database**
  - **Secrets:** The storing and retrieving of data.
  - **Services:** Allows connection by a database connector to facilitate the storage and retrieval of data.

Level 2: Database Connectors

- **Label Database Connector**
  - **Uses:** Label Database
  - **Secrets:** The steps to connect to and manipulate a database.
  - **Services:** Connects to a database and executes pushing and fetching of data.
- **ImageObject Database Connector**
  - **Uses:** ImageObject Database
  - **Secrets:** The steps to connect to and manipulate a database.
  - **Services:** Connects to a database and executes pushing and fetching of data.
- **Labeller Database Connector**
  - **Uses:** Labeller Database
  - **Secrets:** The steps to connect to and manipulate a database.
  - **Services:** Connects to a database and executes pushing and fetching of data.

Level 3: Core Services

- **Label Server**
  - **Uses:** Label Database Connector
  - **Secrets:** The steps to accept and store a label.
  - **Services:** Converts a label JSON object and passes it to be stored in the database.
- **Object Extraction Manager**
  - **Uses:** Label Confidence Service, Object Extraction Service, Labeller Expertise Calculator
  - **Secrets:** The algorithms used to aggregate labels into usable ImageObjects for model training.
  - **Services:** Calls other modules to efficiently compute the most likely set of objects from a set of independent labels.
- **Image Service Manager**
  - **Uses:** Image Selection Service, Image Mask Service, Image Prior Analyzer
  - **Secrets:** The algorithms used to select images to be served to a user.
  - **Services:** Calls other modules to select the next images to be served to a user.

Level 4: Specialized Services

- **Label Confidence Service**
  - **Secrets:** The algorithms used to determine confidence in detected ImageObjects.
  - **Services:** Calculates the confidence of a detected ImageObject.
- **Object Extraction Service**
  - **Secrets:** The algorithms used to extract ImageObjects.
  - **Services:** Calculates the most likely ImageObjects from a set of labels and priors.
- **Image Prior Analyzer**
  - **Secrets:** The algorithms used to calculate ImagePriors.
  - **Services:** Calculates the prior likelihood of a pixel being relevant.
- **Labeller Expertise Calculator**
  - **Secrets:** The algorithms used to calculate Labeller Expertise.
  - **Services:** Calculates the expertise of a labeller in a given class using previous labels.
- **Image Mask Service**
  - **Secrets:** The algorithms used to modify a given image.
  - **Services:** Modifies an image to improve labelling accuracy or efficiency.
- **Image Selection Service**
  - **Secrets:** The algorithms used to select the next image to show a user.
  - **Services:** Selects an image to show to a given user.

Directed Acyclic Graph (DAG)

The following hierarchy is represented as a directed acyclic graph, with arrows showing the *uses* relationship:

- **Foundational Modules** (Label Database, ImageObject Database, Labeller Database)
  - Used by their respective connectors.
- **Database Connectors** (Label Database Connector, ImageObject Database Connector, Labeller Database Connector)

  - Used by core services like Label Server, Object Extraction Manager, and Image Service Manager.

- **Core Services** (Label Server, Object Extraction Manager, Image Service Manager)
  - Use specialized services to provide advanced functionalities.

- **Specialized Services** (Label Confidence Service, Object Extraction Service, Image Prior Analyzer, Labeller Expertise Calculator, Image Mask Service, Image Selection Service)
  - Provide algorithms and computations needed by the core services.

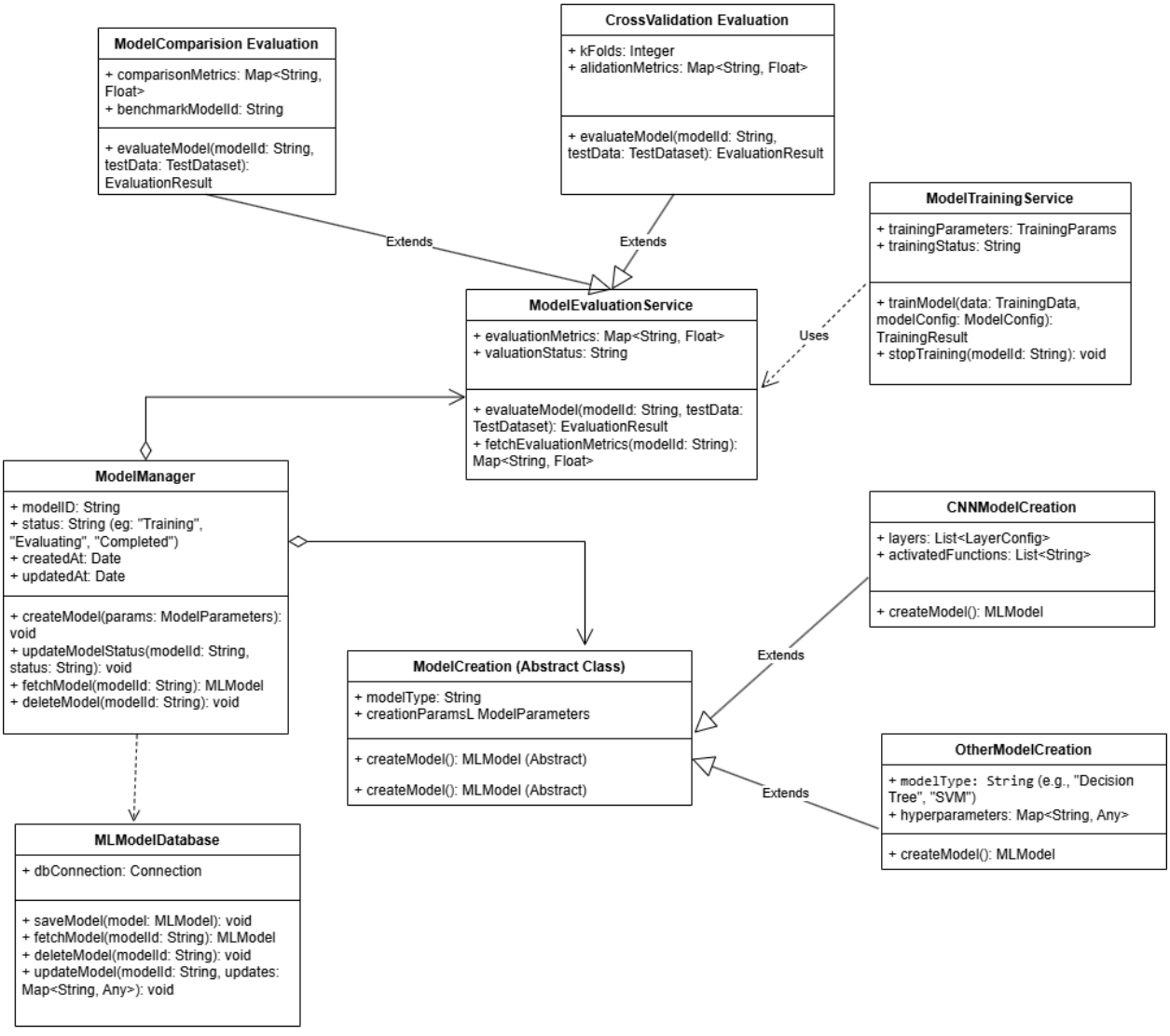

Figure 1: ML Model

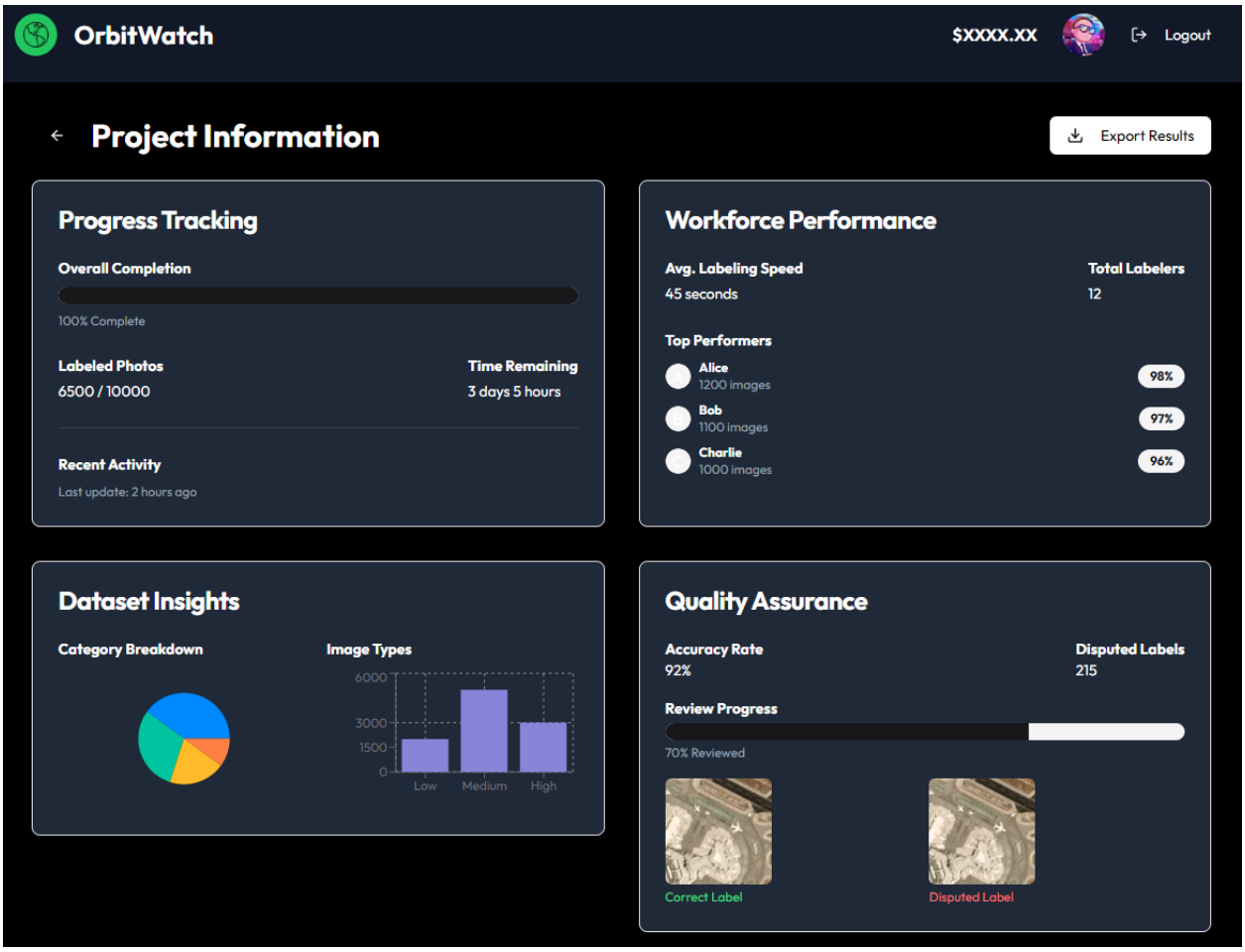

Figure 2: Project selection

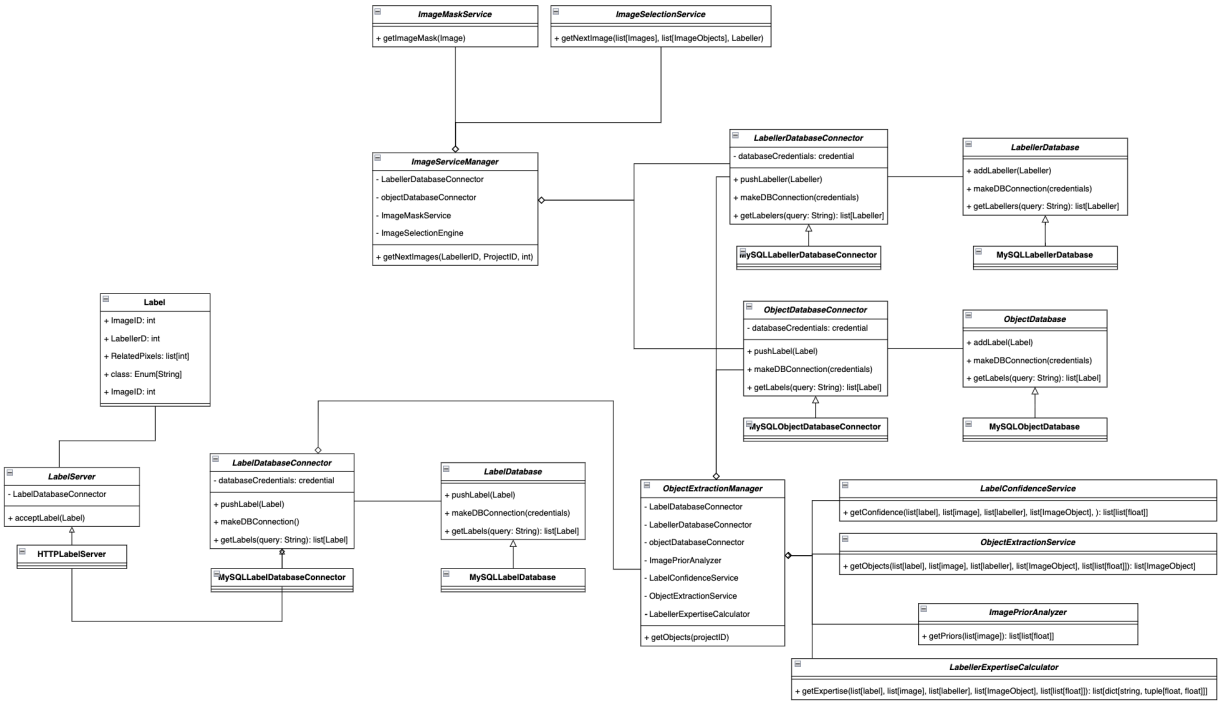

Figure 3: Labelling model

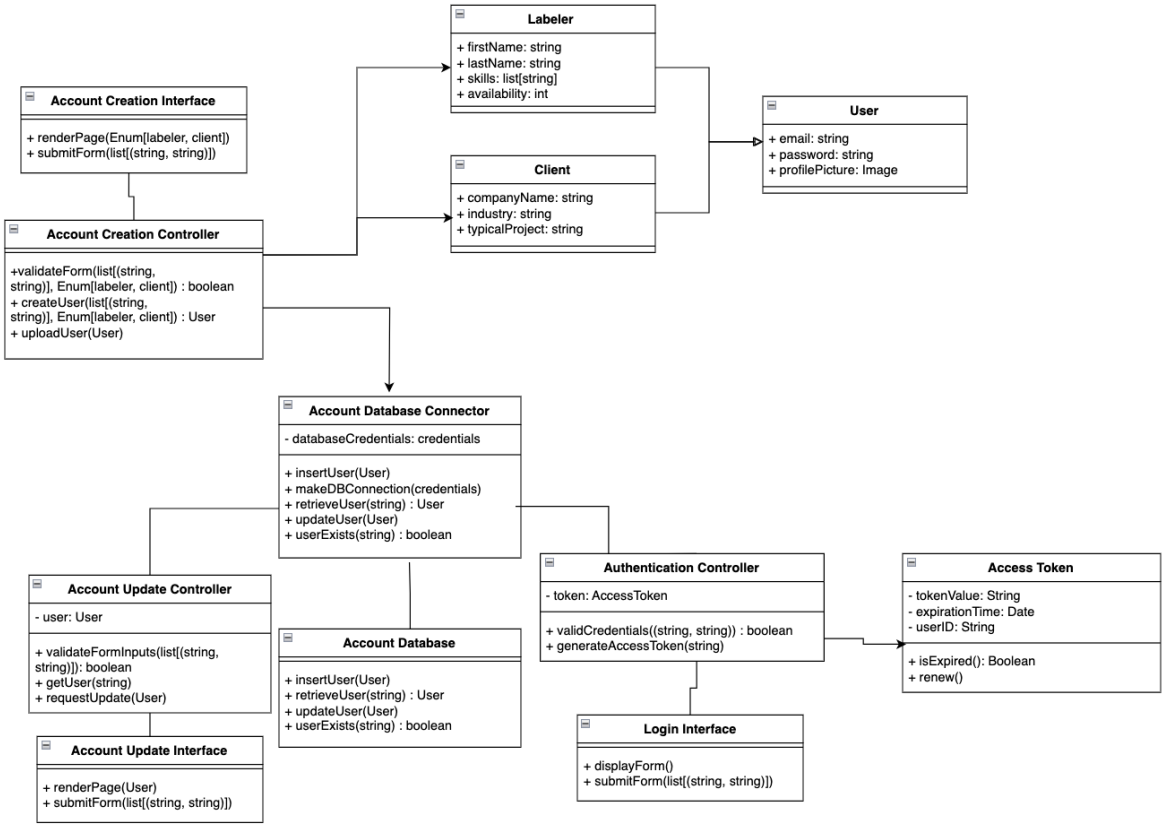

Figure 4: UI interfaces

10 User Interfaces

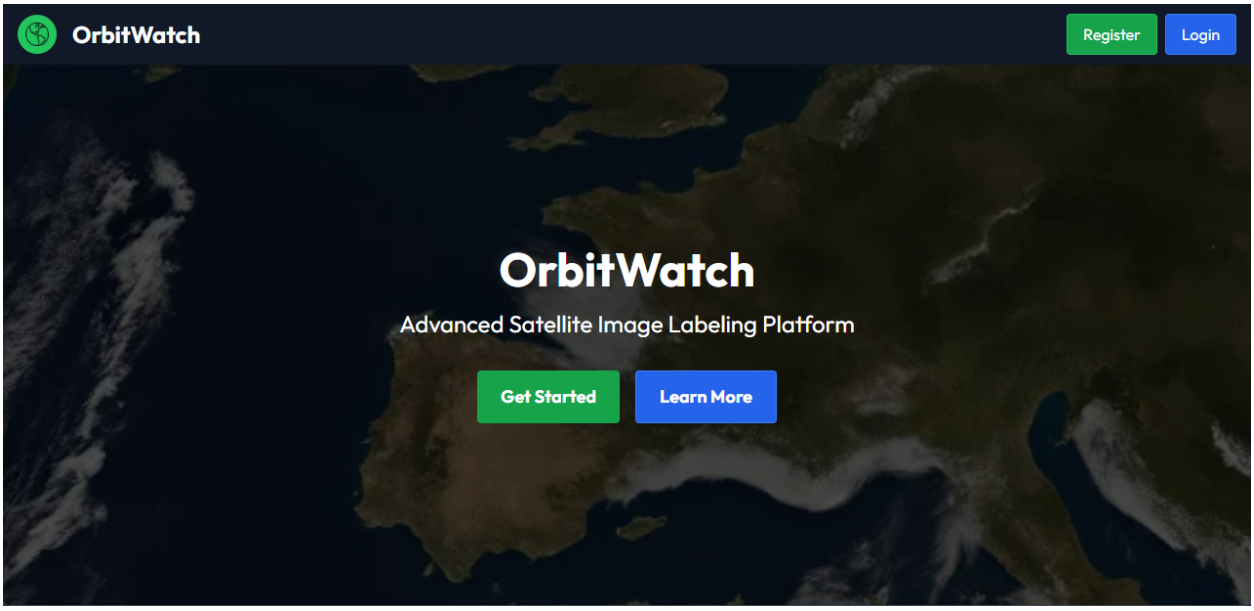

Figure 5: Home Page

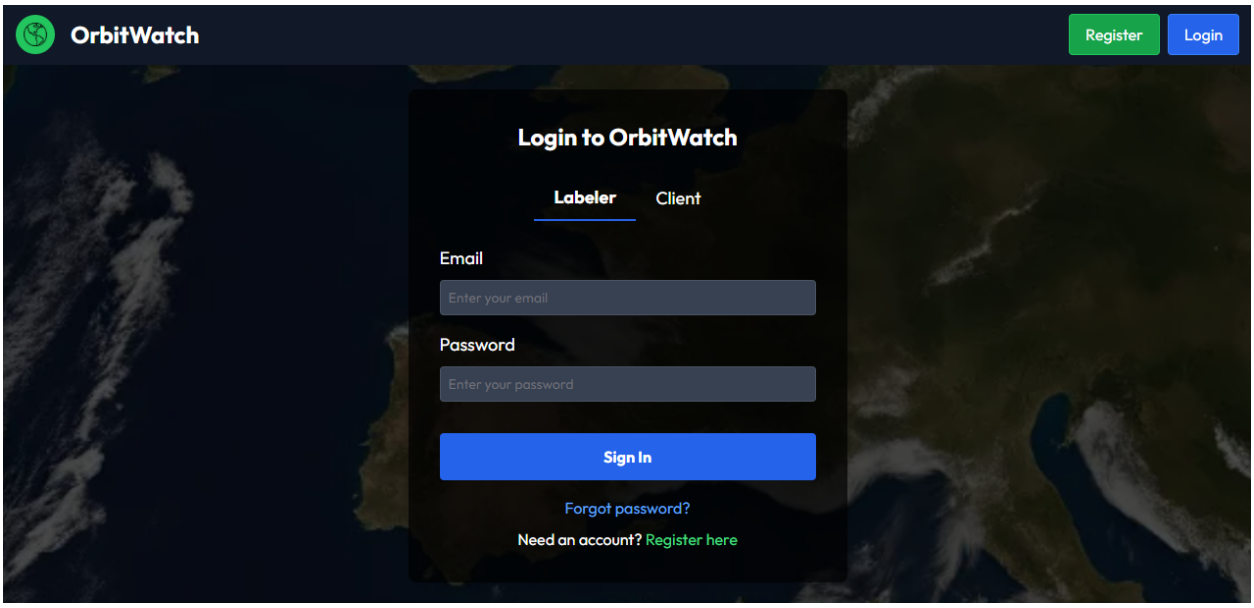

Figure 6: Login Page

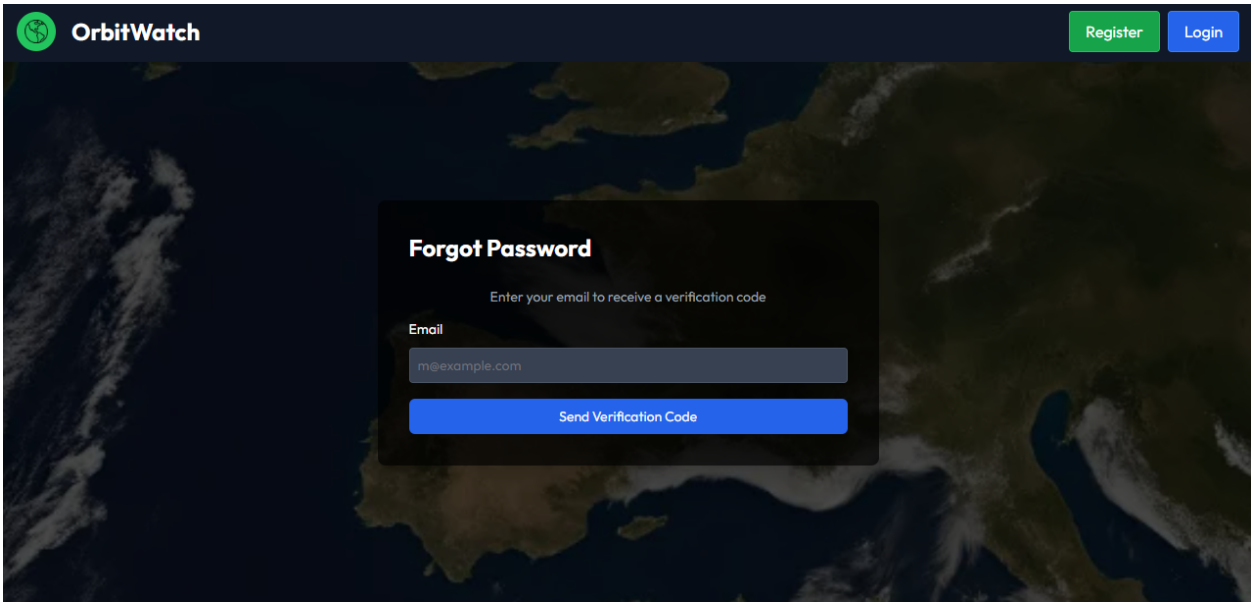

Figure 7: Forgot Password

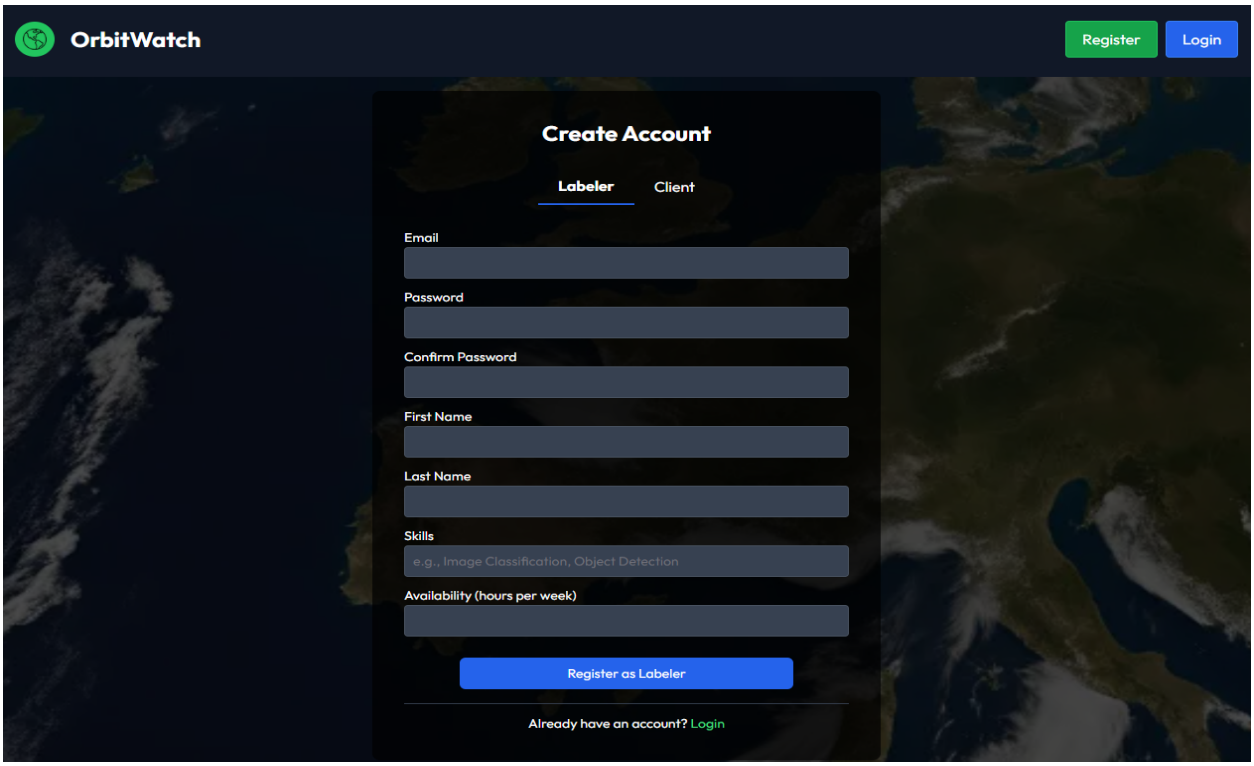

Figure 8: Register Page

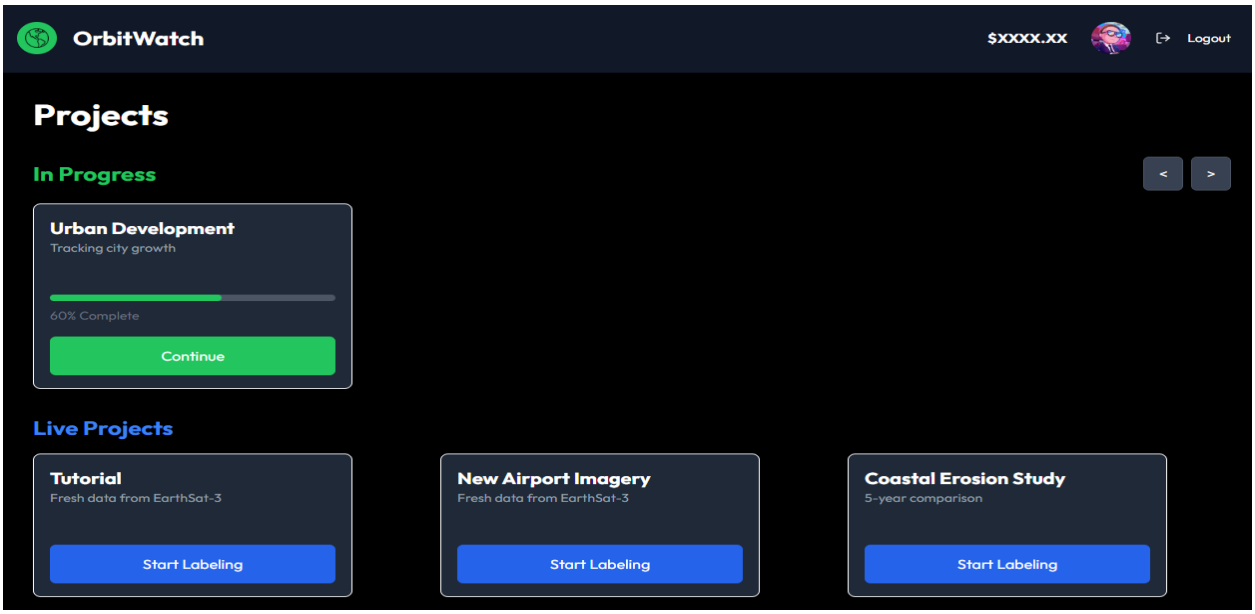

Figure 9: Projects to Label Page

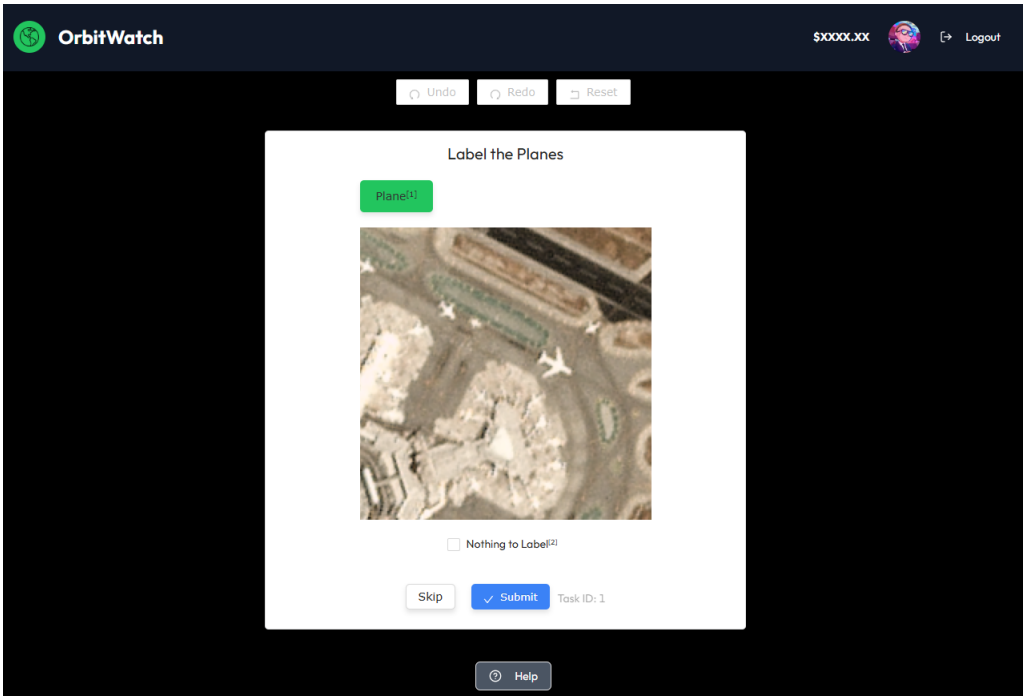

Figure 10: Label Page

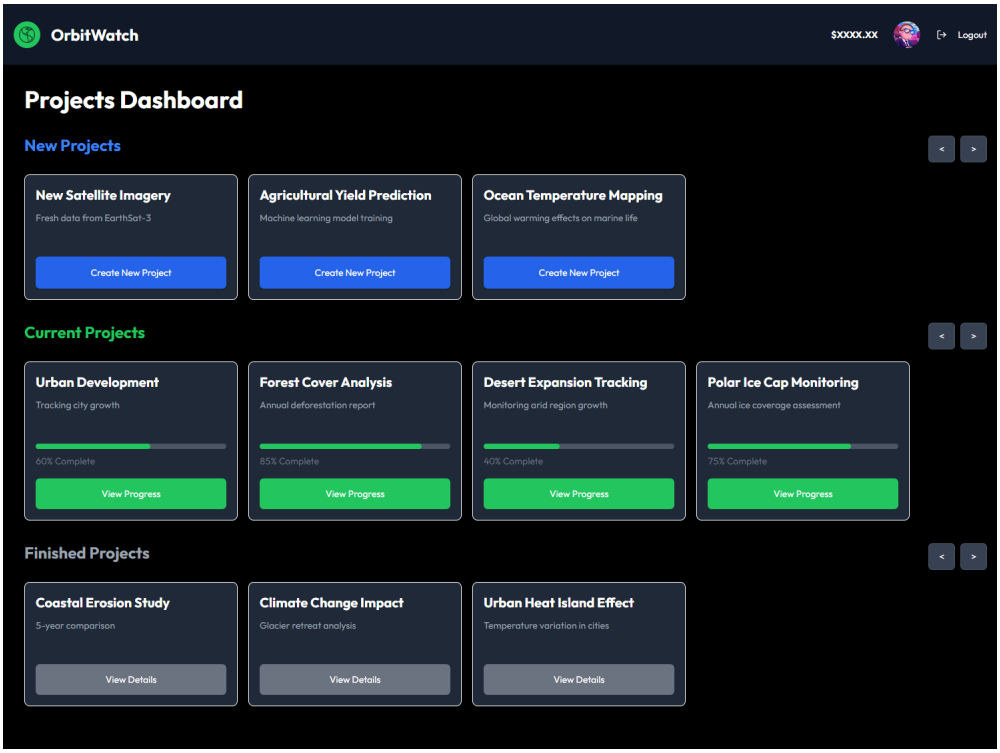

Figure 11: Client Projects Page

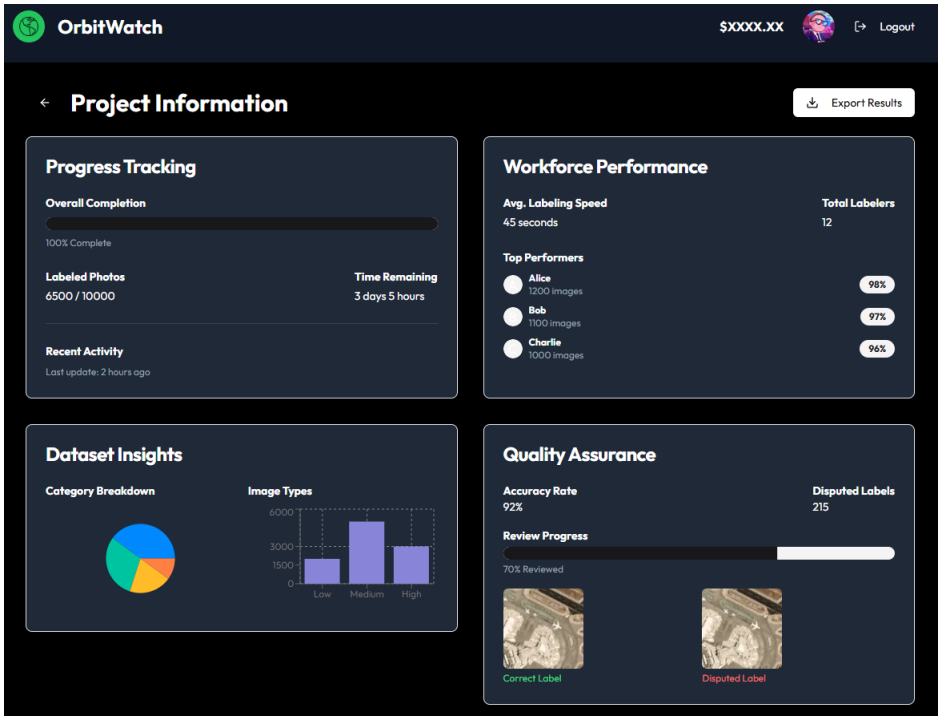

Figure 12: Project Status Page

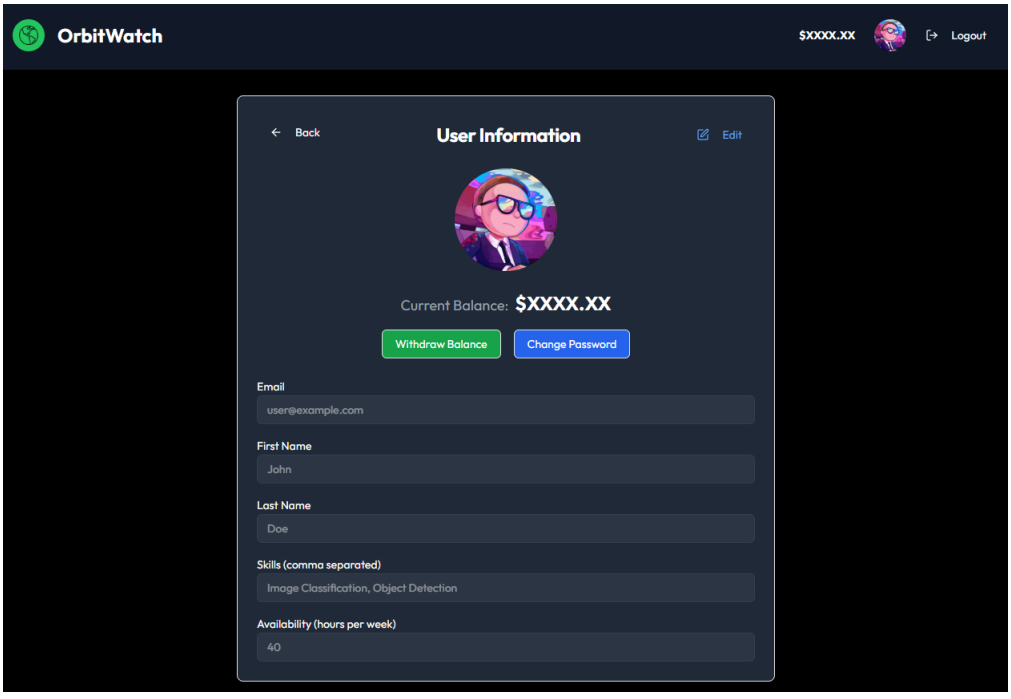

Figure 13: User Information Page

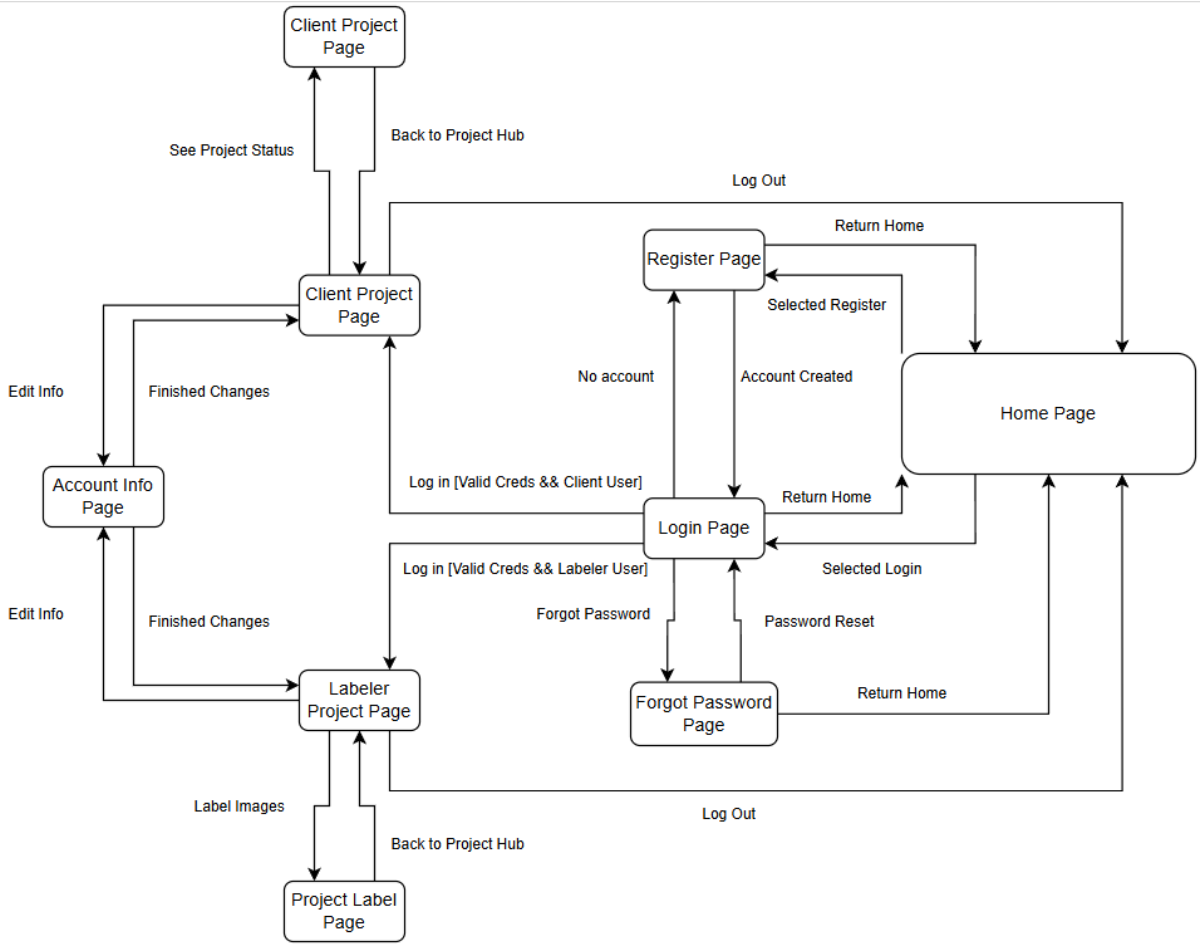

Figure 14: Page Flow Diagram

11 Design of Communication Protocols

Our platform will center around secure, efficient, and scalable communication protocols to facilitate interactions between the frontend, backend, and external services. Below is an outline of the key design considerations:

1. Client-to-Backend Communication:

   HTTPS/RESTful APIs: All communication between the web-based frontend and the backend will occur over HTTPS using RESTful API endpoints. This ensures data in transit is encrypted and secure.

   Authentication & Authorization: We will implement robust authentication mechanisms such as OAuth2 or JWT tokens for verifying and controlling access to API resources.

   Structured Data Format: Requests and responses will primarily use JSON for lightweight, human-readable, and easily parsed messages.

Statelessness: The APIs will be stateless, allowing for easier scaling and handling of parallel requests, critical for enabling multiple users to label the same images concurrently.

Additional Security Measures:

- **Input Validation:** All API endpoints will enforce strict input validation to prevent injection attacks.
- **Database Encryption:** Sensitive data stored in databases will be encrypted at rest using industry-standard encryption.
- **Rate Limiting & Throttling:** To mitigate abuse, API endpoints will implement rate limiting and request throttling.

2. Backend Infrastructure and AWS Communication:

AWS Hosting: The backend services will be deployed on AWS, taking advantage of its scalable infrastructure and regional availability to reduce latency and increase reliability.

Microservices & API Gateway: We may structure the backend as microservices behind an API Gateway to manage routing, authentication, and rate limiting more efficiently.

AWS Database Connectivity: The backend will communicate with AWS-hosted databases (Amazon RDS) via secure connections using AWS SDKs or native drivers over HTTPS or proprietary secured protocols.

- **Secure Credential Management:** Database credentials and API keys will be stored securely using AWS Secrets Manager or IAM roles.
- **Network Isolation:** Databases will be deployed within private subnets, accessible only through strict security group rules.

12 Timeline

Milestone	Deadline	Tasks (Code)	Tasks (Documentation)
Revision 0 Demonstration	Feb 3 – Feb 14	- Implement foundational modules: - **Oleg**: Databases (Label, ImageObject, Labeller) and connectors. - **Kartik and Kyle**: Backend APIs for connectors. - **Mathew**: Frontend for basic database interactions.	Describe the purpose of foundational modules. Write an overview of functionality and initial design.
V&V Report Revision 0	March 7	- Begin **core services** implementation: - **Kartik and Kyle**: Label Server. - **Oleg**: Database and API integration testing. - **Mathew**: Develop basic UI for Label Server interactions.	Document module-level testing strategies. Include high-level testing results. Update SRS and V&V plans.
Final Demonstration (Rev 1)	March 24 – March 30	- Complete **core services**: - **Kartik and Kyle**: Object Extraction Manager, Image Service Manager. - **Oleg**: Optimize database queries for scalability. - **Mathew**: Build advanced frontend for services.	Document core service implementations and integration steps. Update diagrams and traceability matrices.
EXPO Demonstration	April TBD	- Implement **specialized services**: - **Kartik**: Labeller Expertise Calculator. - **Kyle**: Image Mask Service. - **Oleg**: Refine backend for API reliability. - **Mathew**: Enhance frontend interactivity.	Create presentation materials. Include screenshots, results, and use cases for EXPO demo.
Final Documentation (Rev 1)	April 2	- Conduct **end-to-end testing**: - **Kartik and Kyle**: Verify ML pipelines. - **Oleg**: Test backend and database robustness. - **Mathew**: Test UI for edge cases.	Finalize and submit SRS, V&V Plan, and test results. Complete user/admin manuals and deployment guide.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.