



TerraByte

Architectural Requirements

A project for  by  GeekGurusUnion

Demo 2

24 June 2024

Architectural Requirements

[Architectural Design Strategy](#)

[Architectural Strategies](#)

[Architectural Quality Requirements](#)

[Architectural Patterns](#)

[Design Patterns](#)

[Constraints](#)

[Technology Choices](#)

Architectural Design Strategy

Our design strategy focuses on two key approaches: Decomposition Strategy and Quality-Driven Requirements Strategy.

Decomposition Strategy

Concept: This approach involves breaking down the system into smaller, independent components or subsystems, much like constructing a building by dividing it into tasks like laying the foundation, building the walls, and installing the roof. Each component addresses specific aspects of the system, ensuring a modular and manageable architecture.

Benefits:

- **Modularity:** Each component operates independently, enhancing system understanding and management. This modularity simplifies the complexity, allowing developers to focus on individual parts without getting overwhelmed by the entire system.
- **Maintainability:** Changes or fixes can be applied to individual components without affecting the entire system. This isolation of components means that updating or debugging one part doesn't require extensive changes to others, thereby reducing the risk of introducing new bugs.
- **Extensibility:** Adding new features becomes easier as new components can be integrated without disrupting existing ones. This ensures that the system can evolve over time, incorporating new functionalities seamlessly.

Quality-Driven Requirements Strategy

Concept: The design process is guided by the key quality requirements of the system. These requirements define the essential characteristics that ensure the system's success, such as reliability, efficiency, security, and usability.

Benefits:

- **Focus on Quality:** By prioritizing quality from the beginning, the final system is more likely to meet or exceed user expectations. This approach ensures that the design addresses crucial aspects like performance, reliability, and security, resulting in a robust system.
- **Improved User Experience:** The system will be reliable, perform well, be secure, and easy to use. By emphasizing these quality attributes, the system not only fulfills its functional requirements but also provides a pleasant and efficient user experience.

Architectural Strategies

This project aims to develop a user-friendly and reliable crop prediction system for farmers. Our architectural strategy focuses on modularity, quality, and performance by implementing performance tracking, automated testing, and service monitoring. The system prioritizes accessibility for diverse users by offering compatibility across devices, offline access, and a user-centric design, ensuring its adaptability to the challenges faced in agriculture.

1. **Performance**
 - 1.1. **Metric Tracking and Optimization:** Implement continuous monitoring of machine learning (ML) models to gather performance metrics. Utilize these metrics to fine-tune and optimize the models for enhanced performance.
 - 1.2. **Efficient Timing:** Integrate efficient timing mechanisms to log and analyze the time taken by various components of the ML models, identifying areas for improvement.
2. **Testability**
 - 2.1. **Automated Testing Framework:** Establish a comprehensive automated testing framework to ensure rigorous testing across different layers of the architecture.
 - 2.2. **Unit Tests:** Implement unit tests to validate the functionality of individual components.
 - 2.3. **Integration Tests:** Develop integration tests to verify the interactions between different units.
 - 2.4. **End-to-End Tests:** Conduct end-to-end testing to ensure smooth system flow and user experience.
 - 2.5. **Coverage Reports:** Generate and review coverage reports to ensure all critical paths and components are adequately tested.
3. **Efficiency**
 - 3.1. **Service Metrics Monitoring:** Use service metrics, reported by the SaaS provider (e.g., Digital Ocean) or custom-defined in the code, to identify and address bottlenecks.

- 3.2. Performance Optimization: Regularly analyze service measures to optimize system performance and enhance the overall efficiency of the application.
- 4. Compatibility
 - 4.1. Cross-Device Compatibility: Design the application to be compatible with various devices, ensuring seamless performance regardless of system specifications.
 - 4.2. Offline Access: Enable offline access to the app, allowing farmers to view their yield data without needing an internet connection. Implement data synchronization mechanisms to update offline data once the connection is restored.
- 5. Usability
 - 5.1. User-Centric Design: Focus on creating an intuitive and user-friendly interface to enhance the user experience for farmers.
 - 5.2. Accessibility: Ensure the application is accessible to a wide range of users, including those with varying levels of technical proficiency.
 - 5.3. Responsive Design: Implement a responsive design to ensure optimal usability across different screen sizes and orientations.

Architectural Quality Requirements

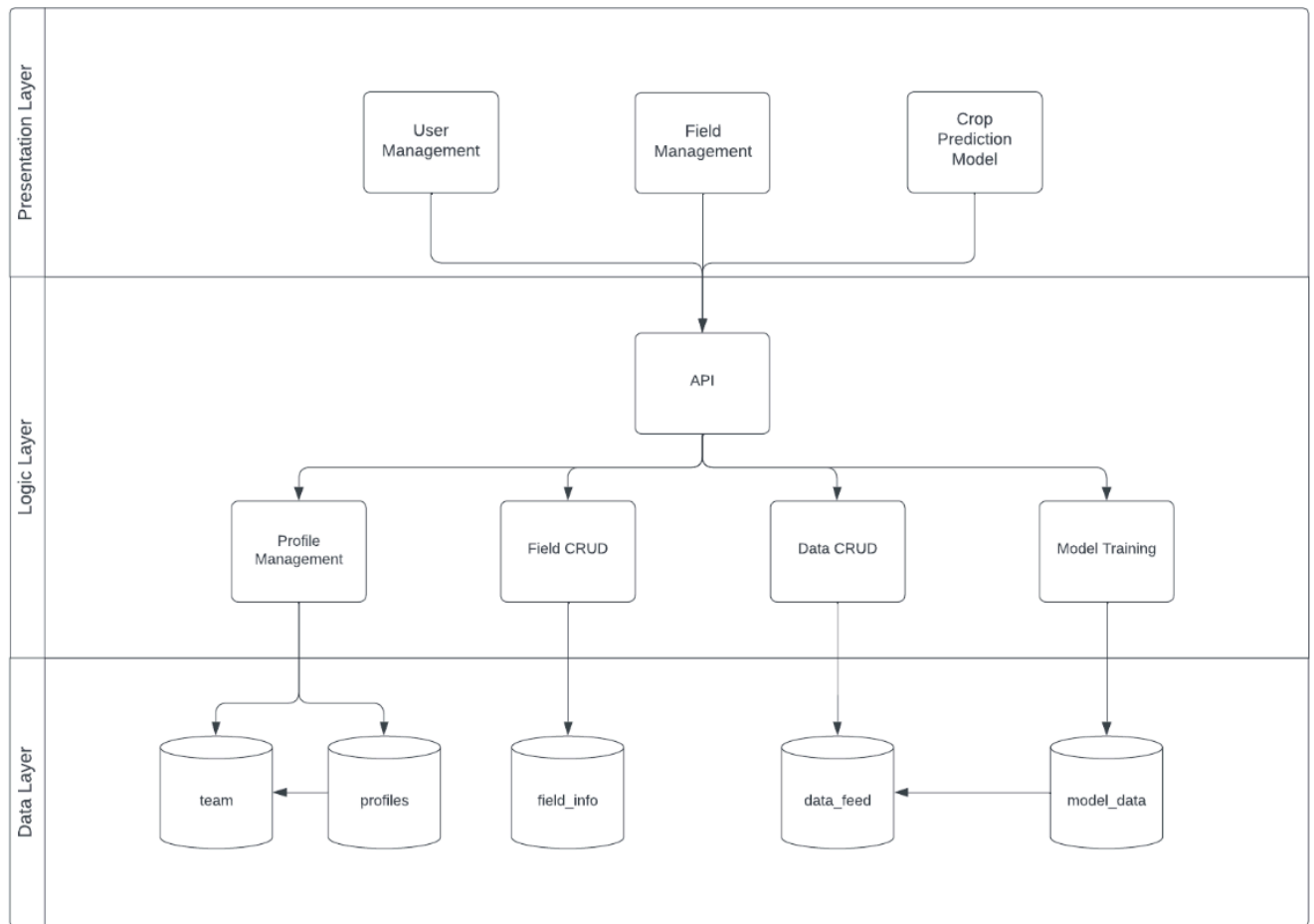
- 1. Performance
 - 1.1. Performance is considered a crucial underlying factor for our system, highlighting the importance of processing power and all-round data analysis capabilities. On the frontend side, effective and rapid rendering is crucial for real time statistics provided to the end user, while being backed by a solid backend system with a rugged API.
 - 1.2. Quantification
 - 1.2.1. ML model training time should be less than 5 minutes.
 - 1.2.2. App loading time should be less than 10 seconds.
 - 1.2.3. API response time should be less than 5 seconds.
- 2. Testability
 - 2.1. Our system's testability measures our test coverage and the ability to place a clear distinction between failing and passing units in our code. This will be required to provide a robust system that works under various circumstances.
 - 2.2. Quantification
 - 2.2.1. Presentation Layer: Make use of Vitest to write unit tests for our frontend.
 - 2.2.2. Logic Layer: Make use of PyTest to write unit tests for our backend.
 - 2.2.3. Data Layer: Make use of PyTest to test the database operations and to ensure consistency in the database to serve accurate and representative data.
 - 2.2.4. Between layers: Make use of Vitest and Postman to test our interactions between layers.

- 2.2.5. System-wide: Make use of Cypress to write end-to-end tests simulating system flow.
 - 2.2.6. 60% coverage overall on Codecov.
- 3. Efficiency
 - 3.1. Because we run sophisticated services in the background on a constrained budget, having a system that avoids wasting efforts, money, and time (i.e. resources) while performing a specific task is crucial.
 - 3.2. Quantification
 - 3.2.1. Graphs provided by our Digital Ocean droplet dashboard (budget, resource utilisation and response time)
 - 3.2.2. Timing function calls and output for our ML model.*
 - 3.2.3. Reporting analytics to a CSV file to analyse bottlenecks.*
- 4. Compatibility
 - 4.1. The app should be designed for easy installation on a variety of devices commonly used by farmers. The app should be configured for automatic updates over WiFi whenever possible.
 - 4.2. Quantification
 - 4.2.1. Make use of a PWA to ensure offline capability.*
 - 4.2.2. Provide a consistent user interface using component libraries.
 - 4.2.3. Fetching new data when connected to the internet by setting up runners that will periodically check for internet connection.*
 - 4.2.4. Saving new data temporarily while the app is not connected to the internet by making use of local storage to temporarily store values while not connected to the internet.*
- 5. Usability
 - 5.1. A system designed to help farmers or agricultural professionals predict the yield and potential success of their crops based on various factors like weather, soil conditions, and historical data. Usability refers to how easy and enjoyable it is for users (in this case, farmers or agricultural professionals) to use the system.
 - 5.2. Quantification
 - 5.2.1. The system should help users achieve their goals (e.g., accurately predicting crop yields) by
 - 5.2.1.1. Providing a clean, intuitive interface with user-friendly navigation.
 - 5.2.1.2. Ensure consistency in the design by using established design systems and component libraries.
 - 5.2.2. The system should be easy and quick to use, allowing users to complete tasks without unnecessary effort or time by providing an easy-to-access help menu within the app for users to get help when needed.*
 - 5.2.3. The system should be user-friendly and enjoyable to interact with, leaving the users feeling satisfied with their experience by
 - 5.2.3.1. Simplify data entry and interaction processes to minimize user effort.

- 5.2.3.2. Provide clear, concise instructions and tooltips to guide users through complex tasks.

Architectural Patterns

Layered Architectural Pattern



Our project will leverage the use of a layered architecture. We firmly believe that a layered architecture will be the most optimal choice for this project with regards to cost efficiency, system effectiveness, and overall performance. Developing these layers independently provides us with the ability to ship a powerful project without having to compromise on hardware and software limitations related to each layer.

Our Layered Architecture Breakdown:

1. Presentation Layer

- 1.1. Functionality: Handles user interactions, facilitates user inputs, and communicates with the logic layer to enable backend functionalities. This layer ensures a seamless user experience and manages all client-side operations.
2. Logic Layer:
 - 2.1. API Gateway:
 - 2.1.1. Functionality: Acts as a single-entry point for the presentation layer to access the logic layer. The API gateway will be used for external API calls, and routing requests based on functionality.
 - 2.2. Core Business Logic:
 - 2.2.1. Functionality: Contains the main application logic, processes user requests, and applies business rules. This includes data processing, computation, CRUD operations, and other core functionalities essential for the application.
 - 2.2.2. Quantification: The Crop Prediction Model is the main model behind AI Crop Prediction, analysing vast datasets to predict crop yields. It considers weather, soil conditions, and historical data to create personalized forecasts for each farm. These predictions empower farmers to optimise resource use and boost their profitability.
3. Data Layer:
 - 3.1. A managed database will store application data in a centralised location, accessible by the logic layer for efficient data sharing and consistency. This layer provides data access mechanisms and interacts with the logic layer to store and retrieve information as needed.

Design Patterns

Singleton

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance. We utilised this pattern to manage global resources or configurations, such as a centralised connection to the database.

Benefits

- Global Access
- Centralised Management
- Resource Management
- Consistency (all components access the same instance of the singleton class)

Factors worth considering

- Alterations to the singleton object not possible (e.g. different parameters).

Facade

With our layered architecture, we ensure a single endpoint to our (backend) system. This ensures that we connect a simple interface with a complex interface (*Facade*, n.d.) without exposing or compromising underlying functionality.

Benefits

- Simplified Interface
- Decoupling (Clients only interact with the facade and are unaware of the internal workings of the subsystems)

Factors worth considering

- Maintenance Overhead (Changes to subsystems may require corresponding changes to the facade)
- Performance Overhead (Additional layer of abstraction may incur a slight performance overhead)

Constraints

Budget Constraints

- Limited financial resources for initial development, deployment, and maintenance.
- Preference for open-source tools and free-tier services to minimise costs.

Hardware Constraints

- Compatibility with common farming hardware, including sensors and IoT devices.
- Limited processing power and memory on devices used by farmers (e.g., older smartphones and tablets).

Network Constraints

- Ensuring functionality in areas with limited or intermittent internet connectivity.
- Efficient data transfer protocols to minimise bandwidth usage.

Technology Choices

1. Frontend

1.1. Vue

- 1.1.1. Vue is a progressive JavaScript framework used for building user interfaces. It is designed to be incrementally adoptable and focuses on the view layer.

1.1.2. Pros:

- 1.1.2.1. Ease of Use: Vue's learning curve is gentle, making it accessible for developers with varying levels of experience.
- 1.1.2.2. Performance: Vue is known for its high performance and efficient rendering.
- 1.1.2.3. Flexibility: It can be used for both simple and complex applications due to its flexible architecture.

1.1.3. Cons:

- 1.1.3.1. Smaller Ecosystem: Compared to React or Angular, Vue's ecosystem is smaller, which might limit the availability of some specialized tools or plugins.

1.2. Nuxt

- 1.2.1. Nuxt.js is a framework built on top of Vue.js that simplifies the development of server-side rendered (SSR) applications and static sites.

1.2.2. Pros:

- 1.2.2.1. Automatic Code Splitting: Nuxt automatically splits the code, which optimizes performance by loading only necessary parts of the application.
- 1.2.2.2. SSR and Static Site Generation: Nuxt supports both SSR and static site generation out of the box, improving performance.

1.2.3. Cons:

- 1.2.3.1. Learning Curve: While built on Vue, Nuxt introduces additional concepts and configurations that require some learning.
- 1.2.3.2. Opinionated Structure: Nuxt enforces a certain project structure, which might not be suitable for every use case.

1.3. TailwindCSS

- 1.3.1. TailwindCSS is a utility-first CSS framework that provides low-level utility classes to build custom designs directly in the markup.

1.3.2. Pros:

- 1.3.2.1. Consistency: Ensures a consistent design system across the application by using predefined classes.
- 1.3.2.2. Utility-First Approach: Provides a wide range of utility classes that eliminate the need for custom CSS, speeding up the styling process.

1.3.3. Cons:

- 1.3.3.1. Verbose HTML: Using utility classes directly in the markup can lead to more verbose HTML.

2. Testing

2.1. Vitest

- 2.1.1. Utilize Vitest for unit testing in the JavaScript ecosystem. This will allow for fast and efficient testing of individual components within the application.

2.2. Cypress

- 2.2.1. Implement Cypress for end-to-end testing to ensure that the entire system works as expected from the user's perspective. This tool will help validate the full application flow and user interactions.
- 2.3. Pytest
 - 2.3.1. Use Pytest for testing Python components, including the machine learning models. This will help in verifying the functionality and performance of the backend logic and algorithms.
- 3. Backend
 - 3.1. PostgreSQL
 - 3.1.1. PostgreSQL is a powerful, open-source relational database management system (RDBMS) known for its robustness, scalability, and standards compliance. It's widely used in many backend systems, including those involving complex data models and high transaction volumes.
 - 3.1.2. **Pros:**
 - 3.1.2.1. Advanced Features: PostgreSQL supports complex queries, foreign keys, triggers, views, and transactional integrity.
 - 3.1.2.2. Extensibility: It allows users to define their own data types, operators, and index types.
 - 3.1.2.3. Performance Optimization: Offers powerful indexing, partitioning, and parallelization features.
 - 3.1.2.4. Community and Support: Large, active community and plenty of documentation and third-party tools.
 - 3.1.2.5. Standards Compliance: Highly compliant with SQL standards, ensuring portability and compatibility.
 - 3.1.3. **Cons:**
 - 3.1.3.1. Complexity: Its vast array of features can be overwhelming and may require a steep learning curve.
 - 3.1.3.2. Resource Intensive: Requires more system resources compared to simpler databases.
 - 3.1.3.3. Setup and Maintenance: Requires careful setup and ongoing maintenance, especially in high-availability and high-transaction environments.
 - 3.2. Python
 - 3.2.1. Python is a versatile programming language widely used in backend systems, including crop prediction applications. It offers robust libraries and frameworks that streamline development and data processing tasks essential for accurate predictions.
 - 3.2.2. **Pros:**
 - 3.2.2.1. Python's extensive libraries such as NumPy and Pandas facilitate complex data manipulation and mathematical computations.
 - 3.2.2.2. FastAPI, a Python library, serves as a lightning-speed API gateway for Python-based backends.

- 3.2.2.3. Python has a big ecosystem with numerous libraries, modules, and community-driven resources.
- 3.2.3. **Cons:**
 - 3.2.3.1. Python is generally efficient for backend tasks, but may struggle with tasks that heavily rely on CPU processing compared to languages closer to the hardware.
 - 3.2.3.2. Python's reliance on external libraries and tools may require additional maintenance efforts (such as keeping requirements.txt up to date).
- 3.3. Tensorflow
 - 3.3.1. TensorFlow, an open-source machine learning framework from Google, is well-suited for building a crop prediction system using LSTM and KNN networks. Its flexibility and scalability enable efficient model development and deployment across different platforms.
 - 3.3.2. **Pros:**
 - 3.3.2.1. TensorFlow is highly flexible and robust, enabling the implementation of complex models like LSTMs.
 - 3.3.2.2. TensorFlow can easily scale to handle large datasets and distribute computation across multiple processing units.
 - 3.3.2.3. TensorFlow is compatible with a lot of languages, in the instance of our system, it fits our use of Python perfectly.
 - 3.3.3. **Cons:**
 - 3.3.3.1. Training large LSTM models can be resource-intensive, requiring substantial computational power and memory. This can be disadvantageous especially for our VM that runs on limited resources.
 - 3.3.3.2. Continuously training a model requires effective resource management to reduce time spent waiting for the training to complete. A pre-trained model would then be loaded to be able to give a “real-time” feeling to the farmer, while training in the background.
- 3.4. Scikit-learn
 - 3.4.1. Scikit-learn is a popular Python library for machine learning that simplifies data preparation. In a crop prediction system, Scikit-learn is used to prepare data before it is fed into Tensorflow for training.
 - 3.4.2. **Pros:**
 - 3.4.2.1. Scikit-learn has an easy-to-use and consistent interface, making it accessible for both beginners and experienced practitioners.
 - 3.4.2.2. The library provides a wide range of preprocessing tools, such as MinMaxScaler for feature scaling, which are essential for preparing data for machine learning models.

- 3.4.2.3. Scikit-learn includes robust tools for model selection and evaluation metrics such as Mean Squared Error (MSE) and R-squared (R^2), simplifying the process of training and assessing models.
- 3.4.3. **Cons:**
 - 3.4.3.1. Scikit-learn is not designed for deep learning. For complex models like LSTMs, TensorFlow would be more appropriate.
 - 3.4.3.2. While efficient, Scikit-learn may struggle with very large datasets or extremely high-dimensional data compared to more specialized libraries.
- 3.5. R
 - 3.5.1. R is a programming language designed for statistical computing and graphics. Its powerful capabilities make it ideal for exploratory data analysis (EDA) and data preprocessing, offering a wide range of tools for manipulating, visualizing, and analyzing data. This prepares data for use in machine learning models.
 - 3.5.2. **Pros:**
 - 3.5.2.1. R comes with a collection of packages for data manipulation, visualization, and statistical analysis, which are essential for EDA and preprocessing.
 - 3.5.2.2. R is specifically designed for statistical analysis, making it an excellent choice for detailed and complex EDA.
 - 3.5.2.3. ggplot2, an R package, provides high-quality, customizable plots, which are crucial for uncovering insights during EDA.
 - 3.5.3. **Cons:**
 - 3.5.3.1. R can be slower compared to some other languages (like Python) for very large datasets and computationally intensive tasks.
 - 3.5.3.2. While powerful, R can have a steeper learning curve, particularly for users who are not familiar with its syntax and functional programming paradigms.
 - 3.5.3.3. R is highly specialized for statistical analysis and data visualization but is less versatile as a general-purpose programming language compared to Python.