Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 12, No. 2, 2025

Research on the Front-End Architecture Design and Visualization Implementation of Agricultural Internet of Things Based on Vue 3

Yaqin Shang, Xiaolin Zhang *, Yang Cao

University of Science and Technology Liaoning, Anshan Liaoning, 114051, China

*** Corresponding author:** Xiaolin Zhang

Abstract: To meet the requirements of real-time data, interface interactivity and system maintainability in agricultural scenarios, this paper proposes and implements an agricultural Internet of Things monitoring platform based on a modern front-end engineering architecture. The front end of the platform adopts the Vue 3 framework and Composition API to construct the responsive view logic, introduces Pinia to achieve modular state management, combines Vite to improve the construction efficiency and performance, and uses ECharts to realize the visual display of device operation status, energy consumption and environmental indicators. In terms of architectural design, this paper divides the system functional modules in detail, builds a clear project directory and component structure, and realizes the decoupling of routing permission control and status. In terms of key technologies, deeply implement responsive data flow, chart component encapsulation and WebSocket real-time communication mechanism to enhance the overall user experience and scalability of the system. This research aims to verify the practical application value of the modern front-end technology system in the agricultural Internet of Things system and provide technical references for the subsequent front-end architecture design of the smart agriculture platform.

Keywords: Smart Agriculture; Front-end Engineering; Vue 3; IoT; Visualization.

1. Introduction

Driven by the technological wave, the traditional agricultural production model is undergoing profound changes and accelerating its transformation towards intelligence and refinement. Smart agriculture, by integrating the entire agricultural production process with advanced technological means, has already become a key path to achieving agricultural modernization. Take agricultural Internet of Things (iot) technology as an example. By densely deploying high-precision sensors, high-definition cameras and intelligent control terminals in vast farmlands, intelligent greenhouses, modern livestock farms and other scenarios, it builds a comprehensive monitoring network. This system can accurately and in real time collect parameters of the crop growth environment, changes in soil moisture, fluctuations in meteorological data, and the operation status of the energy system. It can automatically adjust the production environment based on preset rules, effectively enhancing agricultural production efficiency and resource utilization effectiveness, while significantly reducing human input and production risks [2].

In the smart agriculture ecosystem, the front-end platform plays a crucial role. As a bridge for interaction between users and the system, it undertakes core functions such as data visualization presentation, remote control of equipment, and real-time early warning of abnormal situations. Due to the characteristics of agricultural production scenarios such as scattered equipment, high real-time data requirements, and complex information structures, front-end systems face severe challenges in terms of response speed, smooth interaction, maintainability, and data visualization capabilities. Facing the complex situation where devices are distributed in the fields and farmlands, their states change rapidly, and data types are diverse, how to create an efficient and user-friendly interactive interface has become an important issue in the development of smart agricultural systems. Therefore, designing and building a high-performance, highly scalable and user-friendly front-end architecture has become the core guarantee for the practical application of the agricultural Internet of Things platform.

This research focuses on the modernization and engineering construction of the front-end architecture of the agricultural Internet of Things system, conducting in-depth analysis and practical exploration from multiple aspects such as the architecture design concept, status management mechanism, module organization strategy, data communication scheme, and the realization of visualization technology. Through a systematic review and analysis of the entire front-end development process of the agricultural Internet of Things platform, the aim is to provide referenceable experience and theoretical references for the front-end architecture design and development practice of similar smart agricultural systems.

2. Technical Introduction

(1) Vue 3 and Composition API

In the field of front-end development, Vue.js, with its unique advantages, has become a popular choice for many developers to build user interfaces. As a progressive JavaScript framework, it helps developers efficiently complete development tasks ranging from simple pages to complex applications with its flexible and easy-to-use features. The Vue 3 version launched in 2020 achieved a comprehensive upgrade of functions on the basis of continuing the user-friendly style of Vue 2. The new version not only brings a brand-new Composition API, but also achieves significant breakthroughs in performance optimization, TypeScript integration, etc., greatly enhancing the ability to handle large and complex projects and providing

a broader space for developers to create diversified application scenarios.

The Composition API can be regarded as one of the core highlights of Vue 3. Its emergence provides a brand-new idea for developers to organize and reuse code logic. Compared with the option API adopted in Vue 2 - which divides data, methods, lifecycle hooks, etc. by category into different options, the Composition API breaks this traditional pattern and allows developers to closely combine logic with component states in a more free way. For example, when dealing with a complex function involving data acquisition, status update and side effect operations, using the Composition API can centralize the writing of related code, avoiding the code being scattered among multiple options, thereby making the code structure clearer and more readable. It is particularly suitable for the management and maintenance of complex business logic in large-scale projects.

(2) Pinia state Management

Pinia is the modern state management library officially recommended by Vue 3. It draws on the concepts of Vuex and has the characteristics of being lightweight and flexible. By deeply integrating the responsive API of Vue 3, it realizes state management with an elegant and efficient mechanism. Developers can conveniently access the modified state by using the useStore hook. Combined with the pinia- Plugin-PersistedState plugin, state persistence can also be easily achieved, enhancing the stability of the application and the user experience [5].

(3) ECharts data visualization

In the field of data visualization, ECharts, with its CanCanvas technical architecture, has become a powerful and widely applicable data visualization library. Whether it is the drawing of regular statistical graphs, dynamic real-time graphs, or interactive charts, it can handle them all with ease. In the practical application of agricultural Internet of Things platforms, ECharts plays an irreplaceable role and is widely used in scenarios such as energy consumption trend analysis, battery status monitoring, and complex Sankey chart display. By generalizing and encapsulating the chart components and flexibly passing in dynamic configuration items (Options), developers not only achieve efficient reuse of charts on different pages but also endow charts with the ability to respond and update in real time according to data changes. In addition, ECharts' rich interactive event interfaces provide strong support for the development of interactive functions such as chart linkage operations and click jump links, significantly enhancing the intuitiveness of data presentation and greatly improving the user's interactive experience during the data exploration process [3].

(4) Vite build tool

Relying on the native ES Module mechanism, Vite innovatively breaks the traditional development process. In the development mode, it discards the complicated packaging link and directly realizes the rapid loading of modules. This transformative approach significantly improves the project construction speed and makes the hot update response more agile and efficient. With the powerful dependency pre-packaging capability of ESBuild, the build time of Vite has a significant advantage over the traditional Webpack architecture. In the actual project practice, Vite actively applies on-demand packaging and dynamic chunk separation strategies to precisely optimize the loading speed of the first screen and effectively shorten the waiting time for users. Its powerful plugin mechanism greatly simplifies the import process of Element Plus components, seamlessly integrates components and styles, and significantly improves the development efficiency and project deployment quality.

(5) TypeScript and Engineering Specifications

TypeScript, as an enhanced language of JavaScript, fundamentally improves the readability and maintainability of the code through static type checking and intelligent type inference. In the practice of platform development, it is widely used in core links such as component writing, state declaration and interface encapsulation [1], and is combined with ESLint and Prettier to achieve code quality control. Strict type definitions and interface specifications not only effectively avoid runtime errors but also build a solid security line for team collaborative development and code reconstruction [4].

3. Platform Architecture Design

(1) System function division

The core business function modules of the platform cover multiple key areas: The equipment monitoring module integrates map positioning, alarm presentation and equipment detail query functions by tracking the operation status of environmental sensors and photovoltaic equipment in real time; The energy management module focuses on the display of data such as photovoltaic power generation, power consumption and energy storage status, and conducts energy consumption optimization analysis based on data trends. The data analysis module presents historical data, mean values and fluctuation situations with the help of visual charts, and supports multi-dimensional interactive exploration. The alarm management module summarizes and classifies abnormal alarms in the system, providing the functions of filtering and handling records by time, device, and severity level. The user and permission module realizes user login management, role permission configuration and access control, ensuring the safe and standardized operation of the platform.

(2) Front-end system structure design

To support the core business function modules of the platform, the front-end projects adopt a clear directory structure and modular organization method. The main project directory is as follows:

In the project architecture design, the strategy of "separating business logic from UI components" is adopted: Under the "views/" directory, create independent subdirectories by business module to achieve logical decoupling of different functional sections and avoid code coupling confusion. The "components/" directory serves as a unified repository for UI elements and interaction components. Reusable components like DeviceMap.vue and BaseECharts.vue are stored centrally. It not only improves the reuse efficiency among modules, but also ensures the consistency and standardization of the interaction experience of the entire platform

(3) State management logic design

The front-end system adopts Pinia to build a modular state management system, which is divided into AuthStore, DeviceStore and ConfigStore according to the business dimension. AuthStore manages the user authentication status and permission information, and realizes session persistence through localStorage. DeviceStore is responsible for obtaining device data and updating the status, driving the real-time refresh of the monitoring interface. ConfigStore supports multi-language switching and theme customization.

Components obtain state references through useStore(), follow the principle of unidirectional data flow to avoid direct modifications, and ensure state consistency. Take the change of device status as an example. After the DeviceStore triggers an update, the map annotation, detail card and alarm module synchronously refresh the UI through a responsive mechanism to achieve multi-component collaborative rendering.

```
agriiot-platform/
├── src/
│   ├── api/          # Divide API encapsulation by business
│   ├── assets/       # Static resources (images, styles)
│   ├── components/   # Common reusable components (such as chart cards, map annotations)
│   ├── views/        # Page view components are organized by business modules
│   ├── store/        # Pinia status Management Module
│   ├── router/       # Routing configuration and permission control logic
│   ├── utils/        # General utility functions (such as time formatting, permission handling)
│   └── main.js       # Application entry file
```

Figure 1. Main directory of the project

(4) Routing and permission control design

Front-end routing and access control build a dynamic and controllable system based on Vue Router. Routes are divided into public routes (such as login pages) and restricted routes (such as device management and data analysis). After users log in, accessible route configurations are dynamically generated according to permission roles. The global route guard (router.beforeEach) implements permission verification: unauthenticated users accessing restricted routes are automatically redirected to the login page. If the permission is insufficient, it redirects to the 403 page. When the permission changes, the routes and menus are dynamically updated to ensure real-time consistency. In terms of performance optimization, the routing lazy loading technology is adopted. Components are dynamically imported through import() and combined with Vite's chunk splitting strategy to load page resources on demand, effectively reducing the loading time of the first screen and improving the user experience and system response efficiency.

4. Realization of Key Technologies

(1) Responsive architecture mechanism

The Vue 3 responsive system implements data Proxy based on Proxy and establishes the automatic mapping relationship between data and views by intercepting attribute read and write operations. The platform uses the reactive API to create reactive objects as view data sources for dynamic data such as device status, energy consumption data, and alarm information. When the data changes, it triggers the automatic update of dependent components. Take the device monitoring page as an example. The device status (deviceStatus) serves as the top-level reactive object. Its online status, performance indicators and other attributes are proxyed by the Proxy. When the local map annotation, status card, and alarm module access these attributes, the dependency relationship is recorded. When the device status changes from "online" to "offline", the responsive system automatically notifies dependent components to synchronously update the map color, status ICONS and alarm prompts, achieving declarative programming of "status as view" and avoiding the complexity of traditional DOM operations.

(2) State Management and Component Decoupling (Pinia + Axios)

The platform builds a highly cohesive and low-coupling state management system based on Pinia. Independent Store modules are divided according to the domain model, and State, actions and getters are encapsulated through defineStore. Components use useStore() to obtain instances and storeToRefs() to create reactive references, ensuring the consistency of unidirectional data flow when multiple components share the state. At the data communication layer, the api/ module uniformly encapsulates Axios instances. Tokens are automatically injected through the request interceptor, and the response interceptor handles exceptions such as 401 jumps and 500 error prompts, and verifies the format of the returned data, achieving the decoupling of the front-end and back-end interfaces. Take the device management page as an example. Components trigger data requests by calling the action of the Store. The action internally encapsulates API calls and status update logic, achieving the hierarchical design goal of "data processing cohesion and component logic simplification", and improving system maintainability and development efficiency.

(3) Data Visualization Encapsulation (ECharts module)

This platform builds a data visualization system with ECharts as the core rendering engine, develops the BaseECharts universal component based on the Vue 3 composite API, receives chart configuration and style parameters through props, and realizes responsive layout in combination with ResizeObserver. The component-based encapsulation strategy supports the customization of multiple chart types: visualizing the energy flow path using Sankey diagrams and distinguishing energy types through color coding; The battery status is displayed in real time using the dashboard component, and the percentage data is dynamically rendered through gauge charts. In terms of animation implementation, the device energy consumption line graph applies a smooth transition effect, and the battery health dashboard dynamically updates the pointer Angle through setOption(). All charts are bound to reactive data sources to achieve declarative rendering of "data as graphics". In terms of interaction design, it supports event monitoring such as clicks and hovers, which can trigger pop-up details or page jumps. This encapsulation not only ensures the consistency of display but also provides extension flexibility through the event callback mechanism, effectively meeting the data visualization requirements of different business scenarios.

(4) Real-time data processing mechanism (WebSocket+ polling)

To meet the real-time requirements in the agricultural Internet of Things scenario, the platform combines the WebSocket long connection and the timed polling mechanism. For high-frequency data (such as device status, sudden alarms), the data pushed by the server is received in real time through WebSocket. The front end listens for onmessage events and updates the status according to the pushed content. For low-frequency data (such as daily/weekly energy consumption statistics), setInterval is used to initiate API requests at regular intervals, combined with a manual refresh button to provide update flexibility. To avoid repeatedly requesting data from the same device within a short period of time, the platform uses a Map object to cache the Promise of requests to be processed, and at the same time caches data such as the device list and user permissions in the Pinia persistence plugin, reducing unnecessary requests and state initializations.

5. System Implementation Effect and Evaluation

The platform achieves a high-performance architecture through Vite construction optimization and the responsive mechanism of Vue 3: On-demand packaging splits Element Plus, ECharts, etc. into independent code blocks, reduces the loading volume of the first screen by 30%, and controls the average loading time within 1.5 seconds; In the development stage, HMR is utilized to achieve a 200ms-level component update response and improve the development efficiency. At runtime, based on the Proxy and Pinia state subscription, the component re-rendering delay is less than 50ms, meeting the real-time update requirements of high-frequency data in agricultural monitoring. At the visual interaction level, ECharts charts support data exploration, area zooming, and click jumping. Combined with multi-modal alarm feedback such as floating prompts and map flashing, and with multi-step form processes featuring asynchronous verification, an efficient and convenient operation interface is constructed. The engineering architecture adopts a modular design. Business modules are independently stored in the views/ directory. Dynamic configuration of highly reusable components such as DeviceMap and BaseECharts is achieved through props. The development type safety is guaranteed with the help of TypeScript's full-chain type definition. Internationalization and theme expansion are achieved based on vue-i18n and SCSS variables, supporting the rapid addition of language packs and style themes. Practical verification shows that this architecture provides solid support for the medium and long-term maintenance and functional iteration of the system through logical separation, type constraints and flexible expansion mechanisms.

6. Summary

This paper conducts research on the front-end engineering architecture design and implementation of the agricultural Internet of Things monitoring and management platform. Based on modern front-end technology stacks such as Vue 3, Pinia, Vite and ECharts, a front-end system with a high-performance responsive architecture and engineering system is constructed. This system realizes core functional modules such as agricultural equipment status monitoring, energy dispatching management, data visualization analysis and alarm processing. Through modular design, status management optimization and visualization component encapsulation, it shows significant advantages in maintainability, response efficiency and user experience, effectively verifying the applicability of the modern front-end technology stack in the agricultural Internet of Things scenario. The current system still has room for improvement in aspects such as the optimization of the WebSocket communication mechanism, the performance tuning of complex chart rendering, the improvement of the coverage of component automated testing, and the support for the micro-front-end architecture. The future plan aims to further enhance system performance and scalability through technical paths such as exploring the micro-front-end architecture to achieve module decoupling, integrating WebAssembly and edge computing technologies to improve data processing efficiency, introducing AI algorithms to strengthen data analysis capabilities, and expanding cross-platform deployment solutions. With the acceleration of the digitalization process in agriculture, the core supporting role of front-end technologies in the construction of agricultural information platforms will continue to stand out, and related technological practices and innovations are worthy of further in-depth exploration.

References

[1] Fu Shijun,Lu Songyan,Li Meng, He Zhen, Yuan Jiayang, Bi Chunlan, Liu Li. The Design and Implementation of Smart Agricultural Management System Based on B/S Architecture [J]. Hubei Agricultural Sciences,2025,64(01):154-161.

[2] Xie Zhenhua. Web Architecture Design and Optimal Deployment of Smart Agriculture Sales Platform [J]. Computer and Information Technology,2024,32(05):84-86.

[3] Du Jilong, Li Xinfeng, He Yanfeng, Zhang Wei, Luo Suhua, Cheng Xijie. Design and Implementation of Smart Agriculture System Based on SpringBoot+React [J] Smart Agriculture Guide,2024,4(14):17-20.

[4] Li Hang, Dong Anming, Yu Jiguo, Han Yubing. Design of Smart Agriculture Internet of Things System Based on Front-end and Back-end Separation Architecture [J]. Modern Electronic Technology,2022,45(14):63-68.

[5] Liu Lijun, Zhang Wei, Chen Bo. Web Front-end Design of Smart Agriculture Monitoring System Based on AJAX [J]. Internet of Things Technology,2016,6(01):13-14.