Content

[1 Document Information 3](#_Toc200465610)

[1.1 Document Control Information 3](#_Toc200465611)

[1.2 Revision History 4](#_Toc200465612)

[1.3 Approvals 4](#_Toc200465613)

[1.4 Document Purpose 4](#_Toc200465614)

[1.5 Audience 5](#_Toc200465615)

[2 Project Introduction 5](#_Toc200465616)

[2.1 Project Overview 5](#_Toc200465617)

[2.2 Project Purpose 5](#_Toc200465618)

[2.3 Project Scope 6](#_Toc200465619)

[2.4 Design Goals 7](#_Toc200465620)

[3 Project Architecture 8](#_Toc200465621)

[3.1 Architecture Overview 8](#_Toc200465622)

[3.2 Detailed Architecture Layer 9](#_Toc200465623)

[3.2.1 User Interface Layer 9](#_Toc200465624)

[3.2.2 Security Layer 9](#_Toc200465625)

[3.2.3 Business Logic Layer 10](#_Toc200465626)

[3.2.4 Database Layer 11](#_Toc200465627)

[3.3 Technical Architecture 11](#_Toc200465628)

[3.3.1 Front-end Architecture 12](#_Toc200465629)

[3.3.2 Backend Architecture 12](#_Toc200465630)

[3.3.3 3Database 13](#_Toc200465631)

[3.3.4 Security mechanism (Security) 13](#_Toc200465632)

[4 Class Design 14](#_Toc200465633)

[4.1 Module Classes 14](#_Toc200465634)

[4.2 Subsystem Class Designs 15](#_Toc200465635)

[4.2.1 Authentication Subsystem (auth\_service) 15](#_Toc200465636)

[4.2.2 Data Service Subsystem (data\_service) 19](#_Toc200465637)

[4.2.3 Prediction Subsystem (predict\_service) 20](#_Toc200465638)

[4.2.4 Model Comparison Subsystem (compare\_service) 21](#_Toc200465639)

[4.2.5 Feature Analysis Subsystem (analyze\_service) 23](#_Toc200465640)

[4.2.6 KAN Core Model Subsystem (kan\_core) 24](#_Toc200465641)

[4.3 Class Relationships 26](#_Toc200465642)

[5 Data Design 28](#_Toc200465643)

[5.1 Data Models 28](#_Toc200465644)

[5.2 Database Architecture Design 30](#_Toc200465645)

[6 Module Design 31](#_Toc200465646)

[6.1 User Authentication Module 31](#_Toc200465647)

[6.1.1 Purpose and Functionality 31](#_Toc200465648)

[6.1.2 Module Components 32](#_Toc200465649)

[6.1.3 Data Flow 32](#_Toc200465650)

[6.2 Raster Visualization Module 34](#_Toc200465651)

[6.2.1 Purpose and Functionality 34](#_Toc200465652)

[6.2.2 Module Components 34](#_Toc200465653)

[6.2.3 Data Flow 34](#_Toc200465654)

[6.3 Model Comparison Module 35](#_Toc200465655)

[6.3.1 Purpose and Functionality 35](#_Toc200465656)

[6.3.2 Module Components 35](#_Toc200465657)

[6.3.3 Data Flow 36](#_Toc200465658)

[6.4 Light-Heat Prediction Module 36](#_Toc200465659)

[6.4.1 Purpose and Functionality 36](#_Toc200465660)

[6.4.2 Module Components 36](#_Toc200465661)

[6.4.3 Data Flow 36](#_Toc200465662)

[6.5 Key Factor Analysis Module 37](#_Toc200465663)

[6.5.1 Purpose and Functionality 37](#_Toc200465664)

[6.5.2 Module Components 37](#_Toc200465665)

[6.5.3 Data Flow 38](#_Toc200465666)

[7 Prediction Model Design 38](#_Toc200465667)

[7.1 Data Preprocessing and Model Implementation 38](#_Toc200465668)

[7.1.1 Unified Data Preprocessing Workflow 38](#_Toc200465669)

[7.1.2 Model Implementation 38](#_Toc200465670)

[7.2 Baseline Models 41](#_Toc200465671)

[7.2.1 Random Forest (RF) 41](#_Toc200465672)

[7.2.2 Multi-Layer Perceptron (MLP) 41](#_Toc200465673)

[7.3 KAN Model 42](#_Toc200465674)

[7.3.1 KAN Overview 42](#_Toc200465675)

[7.3.2 Parameter Optimization 42](#_Toc200465676)

[7.3.3 Feature Importance Analysis 43](#_Toc200465677)

[7.3.4 Visualization Capabilities 43](#_Toc200465678)

[8 User Interface Design 44](#_Toc200465679)

[8.1 Page Overview 44](#_Toc200465680)

[8.2 Page functions and interaction details 45](#_Toc200465681)

[8.2.1 Login / Register Page 45](#_Toc200465682)

[8.2.2 Grid Page(Visual heat map) 45](#_Toc200465683)

[8.2.3 Analyze Page(Key Factor Analysis) 45](#_Toc200465684)

[8.2.4 Compare Page(Model comparison) 45](#_Toc200465685)

[8.2.5 Predict Page(Photothermal prediction) 46](#_Toc200465686)

[9 Interface Specification 46](#_Toc200465687)

[9.1 API Design 46](#_Toc200465688)

[9.1.1 API Design Principles 46](#_Toc200465689)

[9.1.2 Authentication 47](#_Toc200465690)

[9.1.3 Core Interfaces 48](#_Toc200465691)

[9.1.4 Detailed Design 50](#_Toc200465692)

[9.2 External Interfaces 55](#_Toc200465693)

[9.2.1 Notification Services (Email Verification Notifications) 55](#_Toc200465694)

[9.3 User Interface to Backend Integration 55](#_Toc200465695)

[9.3.1 Integration Patterns 55](#_Toc200465696)

[9.3.2 Interaction Details 57](#_Toc200465697)

[9.3.3 Typical Interaction Flows 57](#_Toc200465698)

[10 Non-Functional Design 58](#_Toc200465699)

[10.1 Performance Requirements 58](#_Toc200465700)

[10.2 Security Requirements 59](#_Toc200465701)

[10.3 Usability Requirements 59](#_Toc200465702)

[10.4 Maintainability Requirements 60](#_Toc200465703)

[10.5 Compatibility Requirements 60](#_Toc200465704)

1. Document Information
   1. Document Control Information

|  |  |
| --- | --- |
| Document Title | Research and Application of Urban Land Use Change Detection Design Document |
| Version | 1.0 |
| Date | July 10, 2025 |
| Status | Completed |

* 1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Description | Author |
| 1.0 | July 10, 2025 | Complete Initial Design | Project Team |

* 1. Approvals

|  |  |  |
| --- | --- | --- |
| Name | Role | Signature |
| Peng Zhixin | Project Manager |  |
| Wu Honglin | Team Member |  |
| Xu Yangyang | Team Member |  |
| Liu Yang | Team Member |  |

* 1. Document Purpose

This design document aims to define the scope, system architecture, and detailed design aspects of the project titled "Research and Application of Urban Livability in Shanghai Based on Light and Heat Environment using Multi-Source Remote Sensing Data". The purpose of this document is to provide a comprehensive and structured technical foundation for all team members involved in the development, testing, and deployment processes.

* 1. Audience

Research and Development Team

Project Managers

Academic and Institutional Stakeholders

Remote Sensing and Urban Planning Experts

1. Project Introduction
   1. Project Overview

This project aims to provide a platform for the analysis of the solar and thermal aspects of Shanghai's livability analysis. It mainly includes a heat map showing the grid data of Shanghai's solar and thermal aspects, and can be used to identify the key influencing factors of the solar and thermal dependent variables in the Excel data uploaded by users. It also provides solar and thermal predictions using the Kan model, as well as a comparison between the Kan model and other baseline models, aiming to help urban planning personnel analyze the livability level of a certain area in Shanghai through solar and thermal data.

* 1. Project Purpose

The main goal of this project is to build an analytical platform for the livability of Shanghai. The project focuses on two key environmental variables - light and heat, and aims to provide practical suggestions for urban planning and livability improvement. Specifically, the project aims to:

Visualize the spatial distribution of night light radiation and surface temperature in Shanghai in the form of dynamic heat maps based on gridded remote sensing data.

Identify the key influencing factors affecting night light radiation and surface temperature by using statistical correlation on Excel data sets, and support the analysis of user-uploaded data.

Use the KAN model to implement the night light radiation and surface temperature prediction model, so as to enable predictive analysis of environmental conditions at unmeasured time points or future time points.

Compare the KAN model with other baseline models (linear regression, random forest) to verify its effectiveness.

By integrating data visualization, model interpretation and environmental assessment tools into one platform, urban planners and researchers can provide decision support to deeply analyze the livability of specific blocks or regions in Shanghai.

Understand how sunlight and thermal environment affect the overall livability of the city through data-driven, and promote sustainable urban planning.

* 1. Project Scope

This project aims to build a comprehensive data platform for livability analysis in Shanghai, focusing on two core environmental variables: light (nighttime light radiation) and thermal environment (surface temperature). Through remote sensing data processing, visualization and predictive modeling, the platform will provide strong support for urban planning and environmental assessment.

Project Content (In Scope)

Spatial visualization

User data analysis support

Environmental variable prediction modeling

Model effect comparison analysis

Urban livability assessment support

Project Content (Out of Scope)

Does not involve other livability factors other than light and thermal environment (such as air quality, noise, traffic accessibility, etc.);

Does not include mobile phone or mobile application development (this platform is a desktop/web application);

Does not involve cross-city or nationwide promotion and deployment, and the project research area is limited to Shanghai;

* 1. Design Goals

The design goal of this project is to build a Shanghai city livability assessment platform that integrates data analysis, visualization and model prediction in a modular, scalable and data-driven way, focusing on two variables: nighttime light radiation and surface temperature. The platform design will follow the principles of efficiency, interactivity, accuracy and maintainability.

1. Project Architecture
   1. Architecture Overview

System Architecture Layers:

1. User Interaction Layer

Function: Provides a user interface, supports visual analysis, data upload, and model comparison.

2. Security Layer

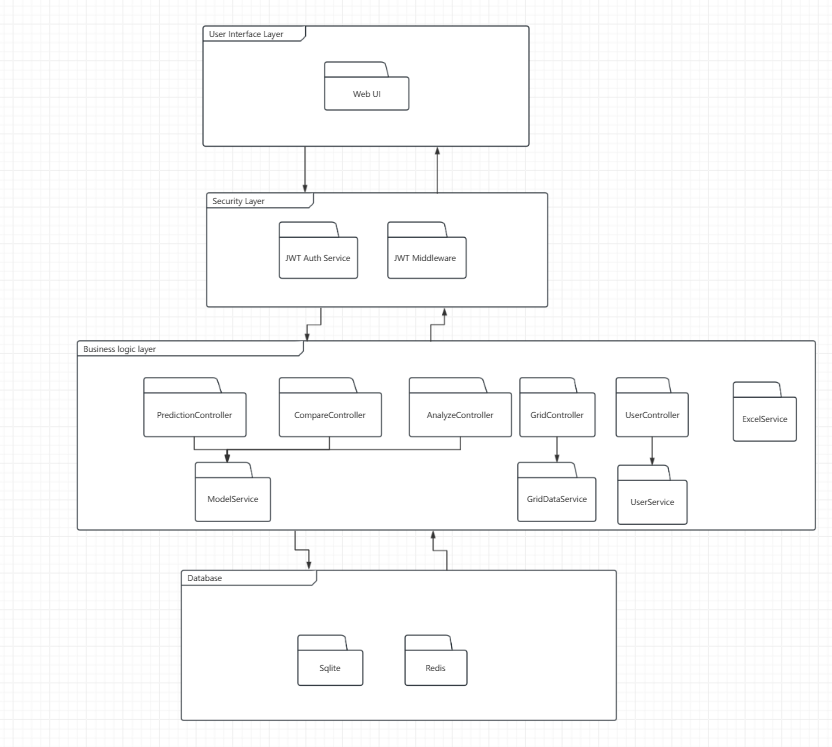
Function: Protects the backend service interface to ensure that only authorized users can access system resources.

3. Application Logic Layer

Function: Responsible for the main business logic processing of the system, including the collaboration between the control layer and the service layer.

4. Data Source

Function: Stores intermediate project data.



* 1. Detailed Architecture Layer
     1. User Interface Layer

The User Interface Layer include:

Web UI: Provides a web-based front-end page to display:

Dynamic heat map of nighttime light intensity and surface temperature in Shanghai;

Excel data results uploaded by users;

Chart comparison of model prediction results; Livability comprehensive score and regional recommendations.

This interface uses a responsive layout to facilitate users to interact with data and download analysis results.

* + 1. Security Layer

This layer is responsible for platform authentication and permission control to ensure data security and user identity legitimacy. Includes:

JWT Auth Service: Provides user identity authentication service based on JWT (JSON Web Token);

JWT Middleware: Used to intercept requests before each protected interface to verify user identity and permissions.

Through this layer, it is ensured that only logged-in and authorized users can access sensitive functions such as analysis, upload, and prediction.

* + 1. Business Logic Layer

This layer is the core logic implementation of the system, responsible for functional modules such as data analysis, model prediction and result generation. The modules are divided as follows:

Controller layer (Controllers):

PredictionController: Calls the KAN model and other comparative models (such as linear regression and random forest) to predict the light and thermal data.

CompareController: Responsible for model evaluation and comparative analysis, supporting performance indicator output.

AnalyzeController: Performs statistical analysis (such as correlation analysis) on the uploaded Excel data to identify influencing factors.

GridController: Provides dynamic heat map services and converts remote sensing grid data into visual layers.

UserController: Handles functions such as user registration, login, and permission management.

Service Layer (Services):

ModelService: Encapsulates the training and reasoning logic of KAN model, linear regression, and random forest;

GridDataService: Processes gridding, interpolation, and dynamic time series management of remote sensing data;

UserService: Processes operations such as user information management and password encryption.

ExcelService: Handles Excel data upload, parsing, and conversion.

This layer decouples modules through dependency injection to achieve functional reuse and flexible expansion.

* + 1. Database Layer

This layer provides persistent storage and fast access services for data, supporting model training, result caching, and user management.

Sqlite: As the main database, it stores user information, analysis results, model configuration, and uploaded structured data;

Redis: As a cache database, it improves the access speed of data such as model prediction results and heat map layers, and reduces latency.

Through the reasonable stratification of the database, the platform supports large-scale data processing and high concurrent access.

* 1. Technical Architecture

This project adopts a front-end and back-end separation architecture, built on a lightweight, high-performance technology stack, and mainly includes four core parts: front-end display layer, back-end service layer, database storage layer and cache layer. The overall architecture is simple and efficient, supporting fast response and data visualization analysis.

* + 1. Front-end Architecture

Technology stack: Vue 3 + Element Plus + Axios + ECharts

Main functions:

Provide a user-friendly Web operation interface;

Realize data upload, prediction request, result display and heat map rendering;

Support responsive layout and interactive animation;

Use Axios to realize API communication with FastAPI backend;

Use ECharts to realize dynamic heat map and chart comparison of night light and surface temperature.

Through component-based development, code reusability and maintainability are improved. At the same time, Element Plus provides a wealth of UI components to ensure interface consistency and development efficiency.

* + 1. Backend Architecture

Technology stack: Python + FastAPI + Uvicorn

Main functions:

Use FastAPI to build high-performance asynchronous API services; Integrate prediction algorithms such as KAN model, linear regression, random forest, etc.;

Provide core functional interfaces including prediction, data analysis, model comparison, etc.;

Organize various business modules through dependency injection: prediction controller, analysis controller, user controller, Excel data processing service, etc.;

Implement user authentication and permission verification based on JWT.

FastAPI supports asynchronous programming, and with Uvicorn deployment, it can significantly improve the response efficiency of model prediction and data processing.

* + 1. 3Database

Relational database: SQLite

SQLite Cache database: Redis

* + 1. Security mechanism (Security)

Authentication: Based on JWT (JSON Web Token)

Implementation method: Issuing Token after successful login;

The front end carries Token through the Header to access the protected interface;

The back-end middleware verifies the validity and permission of the Token.

A stateless authentication mechanism is implemented through JWT to ensure user operation security and system interface access control.

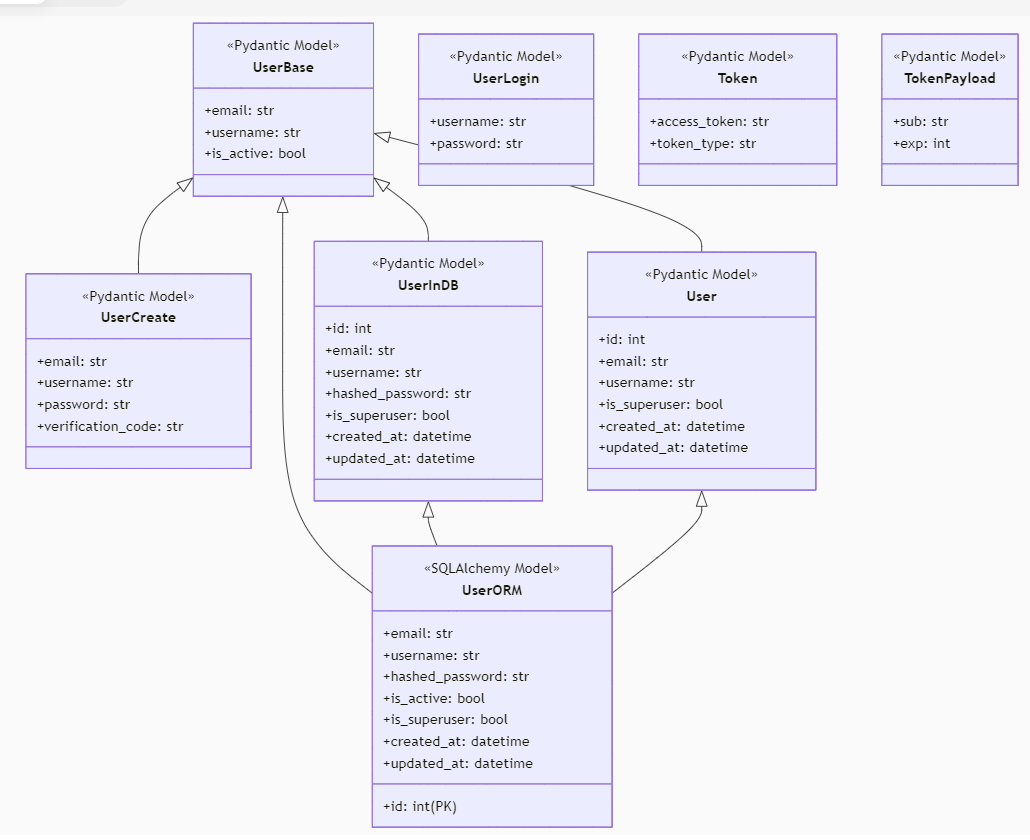
1. Class Design
   1. Module Classes

MultKAN Class

| Attribute | Type | Description |
| --- | --- | --- |
| width | List[int] | Layer dimensions |
| grid | int | Spline grid size |
| k | int | Spline order |
| act\_fun | List[KANLayer] | Activation layers |
| symbolic\_fun | List[Symbolic\_KANLayer] | Symbolic branches |
| feature\_score | np.ndarray | Feature importance |

| Method | Parameters | Returns | Description |
| --- | --- | --- | --- |
| forward | x: Tensor | Tensor | Forward pass |
| fit | dataset, epochs, lr, ... | None | Train model |
| plot | folder: str | None | Save visualizations |
| attribute | None | np.ndarray | Feature importance |
| auto\_symbolic | lib: List[str] | None | Symbolic regression |
| symbolic\_formula | None | Dict | Get expressions |
| saveckpt | path: str | None | Save checkpoint |
| loadckpt | path: str | None | Load checkpoint |

* 1. Subsystem Class Designs
     1. Authentication Subsystem (auth\_service)

****

**UserBase Class**

| Attribute | Type | Description |
| --- | --- | --- |
| email | str | User email address |
| username | str | Unique username |
| is\_active | bool | Account activation status |

**UserCreate Class (extends UserBase)**

| Attribute | Type | Description |
| --- | --- | --- |
| password | str | User password (plaintext) |
| verification\_code | str | Email verification code |

**UserLogin Class**

| Attribute | Type | Description |
| --- | --- | --- |
| username | str | Login username |
| password | str | Login password |

**UserInDB Class (extends UserBase)**

| Attribute | Type | Description |
| --- | --- | --- |
| id | int | Unique user ID |
| hashed\_password | str | BCrypt hashed password |
| is\_superuser | bool | Administrator flag |
| created\_at | datetime | Account creation timestamp |
| updated\_at | datetime | Last update timestamp |

**User Class (extends UserBase)**

| Attribute | Type | Description |
| --- | --- | --- |
| id | int | Unique user ID |
| is\_superuser | bool | Administrator flag |
| created\_at | datetime | Account creation timestamp |
| updated\_at | datetime | Last update timestamp |

**Token Class**

| Attribute | Type | Description |
| --- | --- | --- |
| access\_token | str | JWT token string |
| token\_type | str | Token type (Bearer) |

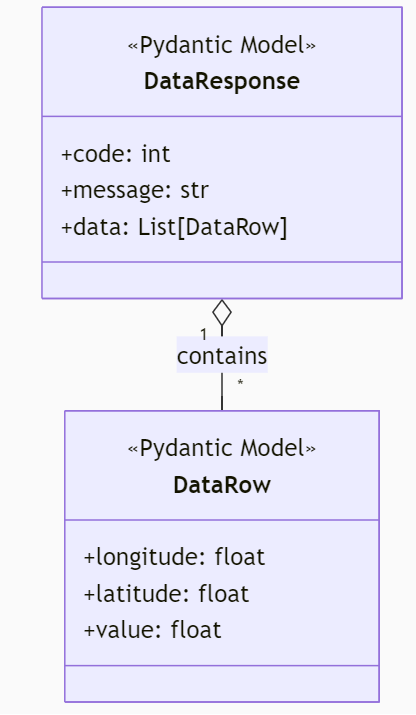
**TokenPayload Class**

| Attribute | Type | Description |
| --- | --- | --- |
| sub | str | Subject (user identifier) |
| exp | int | Expiration timestamp |

**UserORM Class (SQLAlchemy)**

| Attribute | Type | Description |
| --- | --- | --- |
| id | Integer | Primary key |
| email | String | Unique email |
| username | String | Unique username |
| hashed\_password | String | Password hash |
| is\_active | Boolean | Active status |
| is\_superuser | Boolean | Admin status |
| created\_at | DateTime | Creation time |
| updated\_at | DateTime | Update time |

| Method | Parameters | Returns | Description |
| --- | --- | --- | --- |
| verify\_password | plain\_password: str | bool | Password verification |

* + 1. Data Service Subsystem (data\_service)

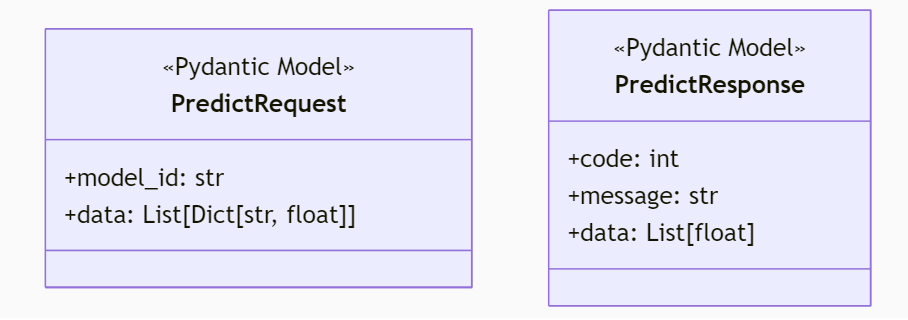
**DataRow Class**

| Attribute | Type | Description |
| --- | --- | --- |
| longitude | float | Geographic coordinate |
| latitude | float | Geographic coordinate |
| value | float | Measurement value |

**DataResponse Class**

| Attribute | Type | Description |
| --- | --- | --- |
| code | int | Status code |
| message | str | Operation message |
| data | List[DataRow] | Data records |

* + 1. Prediction Subsystem (predict\_service)

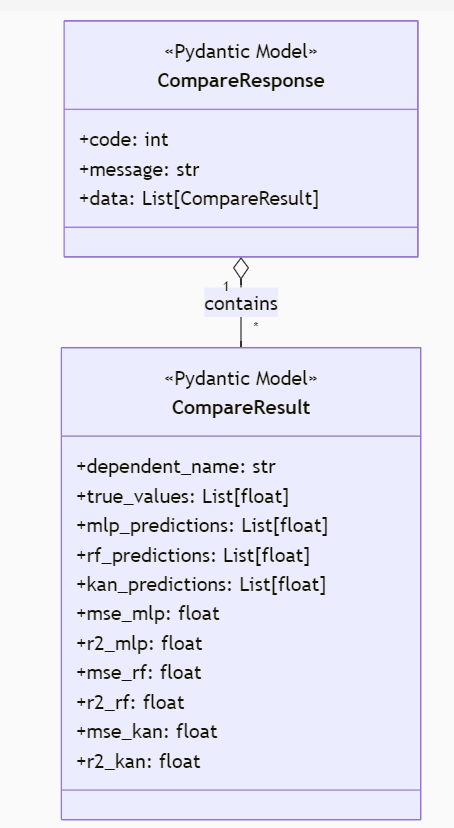


**PredictRequest Class**

| Attribute | Type | Description |
| --- | --- | --- |
| model\_id | str | Target model ID |
| data | List[Dict[str, float]] | Feature values |

**PredictResponse Class**

| Attribute | Type | Description |
| --- | --- | --- |
| code | int | Status code |
| message | str | Operation message |
| data | List[float] | Prediction results |

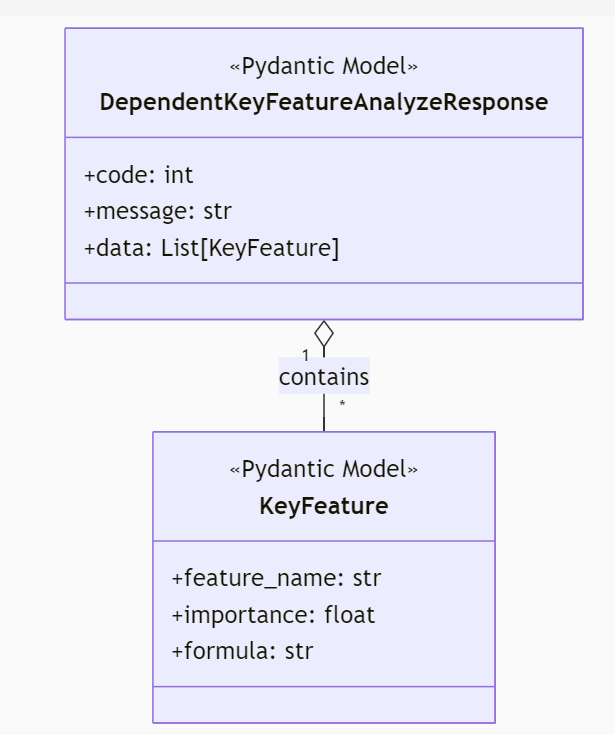
* + 1. Model Comparison Subsystem (compare\_service)

**CompareResult Class**

| Attribute | Type | Description |
| --- | --- | --- |
| dependent\_name | str | Target variable name |
| true\_values | List[float] | Ground truth values |
| mlp\_predictions | List[float] | MLP model predictions |
| rf\_predictions | List[float] | Random Forest predictions |
| kan\_predictions | List[float] | KAN model predictions |
| mse\_mlp | float | MLP mean squared error |
| r2\_mlp | float | MLP R-squared |
| mse\_rf | float | RF mean squared error |
| r2\_rf | float | RF R-squared |
| mse\_kan | float | KAN mean squared error |
| r2\_kan | float | KAN R-squared |

**CompareResponse Class**

| Attribute | Type | Description |
| --- | --- | --- |
| code | int | Status code |
| message | str | Operation message |
| data | List[CompareResult] | Comparison results |

* + 1. Feature Analysis Subsystem (analyze\_service)

**KeyFeature Class**

| Attribute | Type | Description |
| --- | --- | --- |
| feature\_name | str | Input feature name |
| importance | float | Importance score (0-1) |
| formula | str | Symbolic expression |

**DependentKeyFeatureAnalyzeResponse Class**

| Attribute | Type | Description |
| --- | --- | --- |
| code | int | Status code |
| message | str | Operation message |
| data | List[KeyFeature] | Key features |

* + 1. KAN Core Model Subsystem (kan\_core)

**MultKAN Class**

| Attribute | Type | Description |
| --- | --- | --- |
| width | List[int] | Layer dimensions |
| grid | int | Spline grid size |
| k | int | Spline order |
| act\_fun | List[KANLayer] | Activation layers |
| symbolic\_fun | List[Symbolic\_KANLayer] | Symbolic branches |
| feature\_score | np.ndarray | Feature importance |

| Method | Parameters | Returns | Description |
| --- | --- | --- | --- |
| forward | x: Tensor | Tensor | Forward pass |
| fit | dataset, epochs, lr, ... | None | Train model |
| plot | folder: str | None | Save visualizations |
| attribute | None | np.ndarray | Feature importance |
| auto\_symbolic | lib: List[str] | None | Symbolic regression |
| symbolic\_formula | None | Dict | Get expressions |
| saveckpt | path: str | None | Save checkpoint |
| loadckpt | path: str | None | Load checkpoint |

**KANLayer Class**

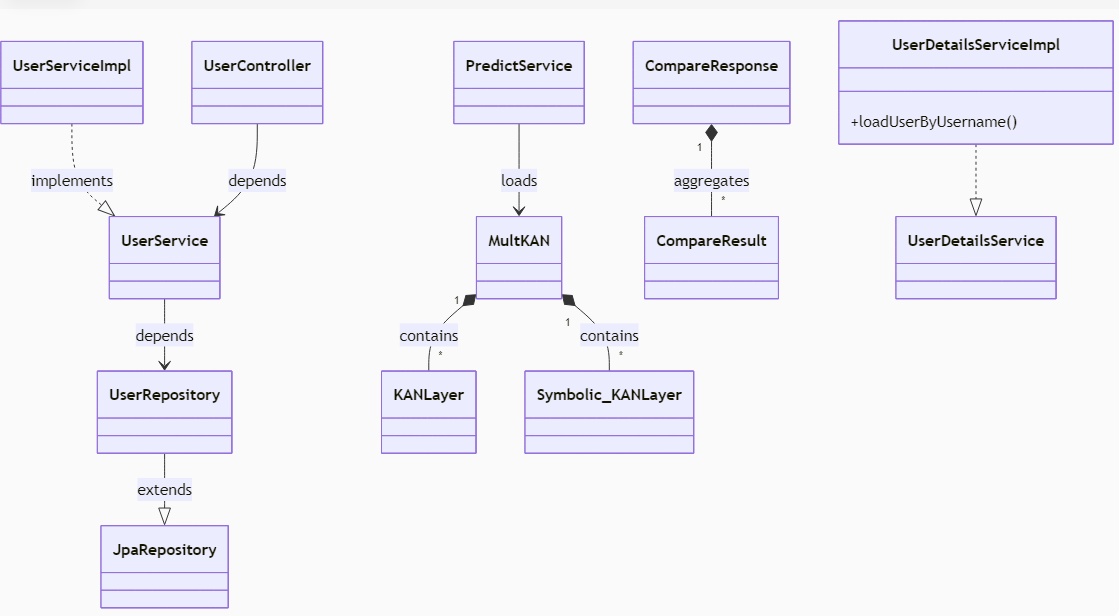
| Attribute | Type | Description |
| --- | --- | --- |
| in\_dim | int | Input dimension |
| out\_dim | int | Output dimension |
| spline\_weight | Tensor | Learnable parameters |

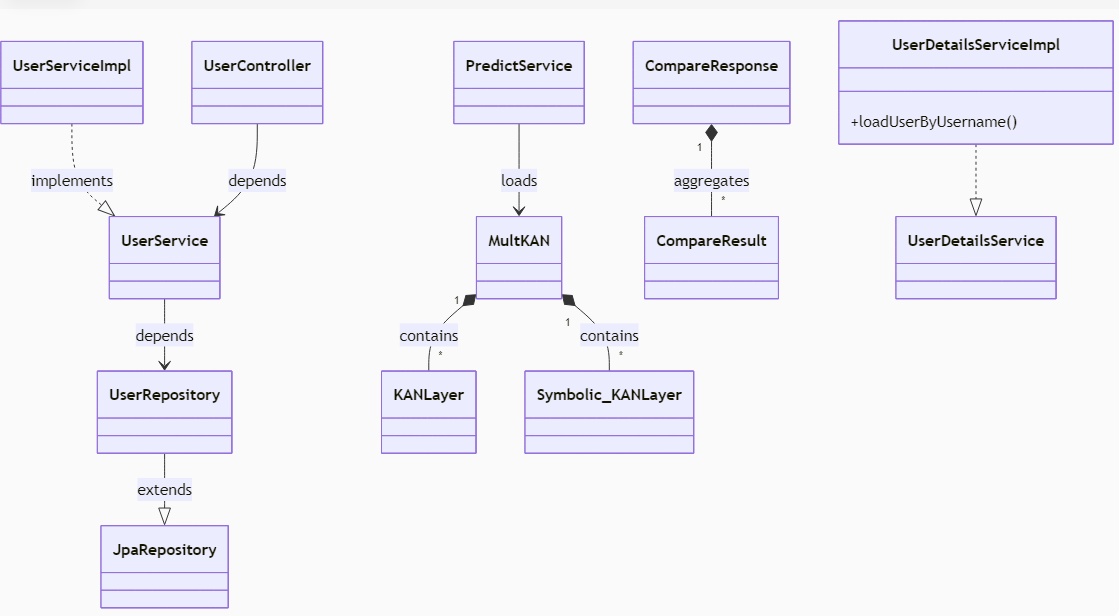
| Method | Parameters | Returns | Description |
| --- | --- | --- | --- |
| forward | x: Tensor | Tensor | Layer forward pass |
| update\_grid | new\_grid: int | None | Update grid size |

**Symbolic\_KANLayer Class**

| Attribute | Type | Description |
| --- | --- | --- |
| in\_dim | int | Input dimension |
| out\_dim | int | Output dimension |
| funs | List[Symbolic\_Layer] | Symbolic functions |

| Method | Parameters | Returns | Description |
| --- | --- | --- | --- |
| forward | x: Tensor | Tensor | Symbolic forward pass |
| formula | None | List[str] | Get symbolic formulas |

* 1. Class Relationships

****The KAN-BackEnd system follows a clean object-oriented design with three core relationship types:

1. Inheritance Relationships

• Services implement interfaces (e.g., UserServiceImpl → UserService)

• Repositories extend JpaRepository

• Pydantic models extend base schemas (e.g., UserCreate → UserBase)

1. Composition Relationships

• MultKAN aggregates KANLayer and Symbolic\_KANLayer

• Responses contain result objects (e.g., CompareResponse → CompareResult)

• User entities own multiple Model records

1. Dependency Relationships

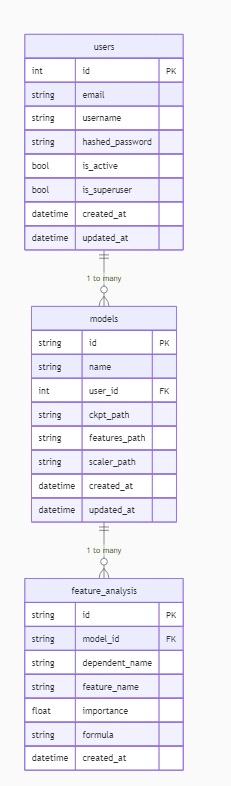
• Controllers → Services → Repositories (layered architecture)

• PredictService depends on MultKAN for model operations

• FeatureService uses Symbolic\_KANLayer for analysis

The design enables evolutionary architecture while ensuring maintainability through well-defined relationship patterns.

1. Data Design
   1. Data Models



**users Table**

| Column | Type | Description |
| --- | --- | --- |
| id | INTEGER | Primary key |
| email | VARCHAR | Unique email |
| username | VARCHAR | Unique username |
| hashed\_password | VARCHAR | Password hash |
| is\_active | BOOLEAN | Account status |
| is\_superuser | BOOLEAN | Admin flag |
| created\_at | TIMESTAMP | Creation time |
| updated\_at | TIMESTAMP | Update time |

**models Table**

| Column | Type | Description |
| --- | --- | --- |
| id | VARCHAR | Model UUID |
| name | VARCHAR | Model name |
| user\_id | INTEGER | Owner user ID |
| ckpt\_path | VARCHAR | Model checkpoint path |
| features\_path | VARCHAR | Feature names path |
| scaler\_path | VARCHAR | Scaler path |
| created\_at | TIMESTAMP | Creation time |
| updated\_at | TIMESTAMP | Update time |

**feature\_analysis Table**

| Column | Type | Description |
| --- | --- | --- |
| id | VARCHAR | Analysis UUID |
| model\_id | VARCHAR | Related model ID |
| dependent\_name | VARCHAR | Target variable |
| feature\_name | VARCHAR | Input feature |
| importance | FLOAT | Importance score |
| formula | TEXT | Symbolic expression |
| created\_at | TIMESTAMP | Creation time |

* 1. Database Architecture Design

The system employs a multi-model database architecture optimized for different data access patterns:

1. Relational Database (PostgreSQL)

• Stores structured application data including users, model metadata, and analysis results

• Implements ACID transactions for critical operations

• Normalized schema design with foreign key relationships

• Optimized indexes for frequent query patterns (user lookups, model searches)

• Supports complex joins for reporting and analytics

1. Time-Series Database (TimescaleDB)

• Stores high-frequency prediction results and model performance metrics

• Time-partitioned tables for efficient historical data management

• Continuous aggregates for pre-computed statistics

• Specialized time-series functions for trend analysis

1. Caching System (Redis)

• Caches model inference results with TTL expiration

• Stores user sessions and authentication tokens

• Implements rate limiting for API endpoints

• Pub/sub channels for real-time prediction updates

1. Object Storage (MinIO/S3)

• Stores serialized KAN model checkpoints (.pth files)

• Maintains feature importance visualizations and analysis reports

• Archives prediction datasets and training logs

• Versioned storage for model artifacts

1. Module Design
   1. User Authentication Module
      1. Purpose and Functionality

The User Authentication Module handles user registration, login, and identity verification. It will:  
• Enable new users to register securely  
• Authenticate existing users via username and password  
• Maintain session control and logout functionality

* + 1. Module Components

**Registration Component**• Functionality: Registers new users into the system  
• Interfaces:

– create\_user\_with\_verification(db , user\_in )

– create\_user(db , user\_in )

**Login Component**• Functionality: Authenticates user credentials and initiates sessions  
• Interfaces:

– get\_user(db , id)

– get\_user\_by\_email(db , email)

– get\_user\_by\_username(db , username)

– authenticate\_user(db , username\_or\_email , password )

* + 1. Data Flow

1. User submits registration/login info via frontend
2. Backend verifies and interacts with user database
3. Authentication token is issued and stored in session
4. Token is used for secure route access and operations

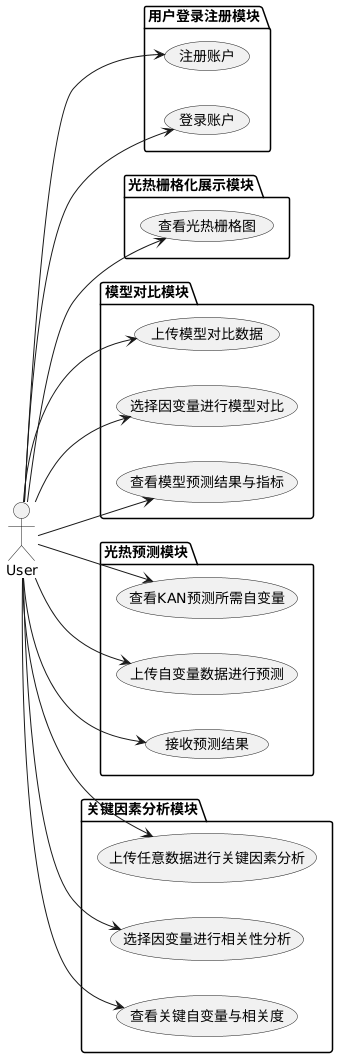


Figure 6.1: Use Case Diagram 1

* 1. Raster Visualization Module
     1. Purpose and Functionality

The Raster Visualization Module displays Shanghai's gridded thermal or light data on an interactive map. It allows selection from:  
• Daytime Heat  
• Nighttime Heat  
• Nighttime Light Intensity

* + 1. Module Components

**Data Selector**  
• Functionality: Allows users to choose the raster data category  
• Interfaces:  
 – selection()

**Map Renderer**  
• Functionality: Renders selected raster data on the Shanghai map grid  
• Interfaces:  
 –updateCharts(variable)

**Data Fetcher**  
• Functionality: Communicates with backend to fetch raster data  
• Interfaces:  
 – load\_data\_by\_target(target , file\_path)

* + 1. Data Flow
* User selects a data category

Frontend requests corresponding raster data from backend

Raster data is rendered as a heatmap on the grid-based map

* 1. Model Comparison Module
     1. Purpose and Functionality

The Model Comparison Module allows users to upload data and compare predictive performance of MLP, RF, and KAN models on selected dependent variables. It will:  
• Accept Excel files in a defined format  
• Let users select one dependent variable (from Daytime Heat, Nighttime Heat, Nighttime Light)  
• Generate predictions using MLP, RF, and KAN  
• Display MSE and R² comparisons

* + 1. Module Components
* **File Upload Interface**  
  • Functionality: Uploads dataset to server  
  • Interfaces:  
   – handleFile(file)

**Model Runner**  
• Functionality: Executes three models on selected target  
• Interfaces:  
 – analyze\_and\_predict(df , target\_dependent\_var)

**Metrics Comparator**  
• Functionality: Computes and displays evaluation metrics  
• Interfaces:  
 –updateCharts(variable)

* + 1. Data Flow
* User uploads Excel file and selects a dependent variable

Backend preprocesses data and runs MLP, RF, KAN

Model outputs and metrics are returned and visualized

* 1. Light-Heat Prediction Module
     1. Purpose and Functionality

This module uses KAN to predict any of the three target variables based on important features. It will:  
• Show static lists of key input features and their importance per target  
• Allow uploading of new data for prediction  
• Return predicted values using KAN

* + 1. Module Components
* **Feature Importance Viewer**  
  • Functionality: Displays top features with correlation for each target  
  • Interfaces:  
   –Display in static form

**Prediction Interface**  
• Functionality: Uploads Excel and receives KAN prediction  
• Interfaces:  
 –predict\_from\_excel(df , target\_dependent\_var)

* + 1. Data Flow
* User selects target and views required features

User uploads new data containing those features

Backend performs prediction and sends back results

* 1. Key Factor Analysis Module
     1. Purpose and Functionality

The Key Factor Analysis Module identifies and displays the most relevant independent variables related to a user-selected dependent variable. It will:  
• Accept Excel files in specified formats  
• Allow user to select one dependent variable  
• Analyze variable correlations  
• Display top influencing factors and correlation scores

* + 1. Module Components
* **Data Input and Target Selection**  
  • Functionality: Accepts Excel and selects dependent variable  
  • Interfaces:  
   – handleFile(file)  
   – selection()

**Correlation Analyzer**  
• Functionality: Identifies significant predictors for the target  
• Interfaces:  
 – analyze\_key\_features\_from\_df(Df , Target , n\_epochs , hidden\_dim , Grid , K , lr)

**Result Display**  
• Functionality: Displays key variables and their strengths  
• Interfaces:  
 – updateCharts(data)

* + 1. Data Flow
* User uploads data and chooses a dependent variable

Backend computes feature correlations

* Results are ranked and shown with visuals

1. Prediction Model Design
   1. Data Preprocessing and Model Implementation
      1. Unified Data Preprocessing Workflow

All models adopt a unified data preprocessing procedure. Key steps include:

* + Data cleaning and handling missing values
  + Feature standardization (e.g., Z-score normalization)
  + Categorical variable encoding (e.g., one-hot encoding)
  + Data partitioning into training, validation, and test sets
    1. Model Implementation

For the three dependent variables — daytime land surface average temperature, nighttime land surface average temperature, and diurnal temperature range — three separate models are developed to perform single-task predictions, forming a comprehensive evaluation framework based on 10-fold nested cross-validation.

According to standard 10-fold nested cross-validation theory, the training and testing of models are conducted using this 10-fold structure, which consists of two components: the inner loop and the outer loop.

**Outer Loop (Model Evaluation)**

* + The dataset is divided into 10 folds.
  + In each iteration, one fold is selected as the test set, while the remaining nine folds constitute the training + validation set.
  + This process is repeated 10 times, ensuring that each fold serves once as the test set.
  + The primary objective of the outer loop is to provide an unbiased performance estimate of the model after hyperparameter tuning.

**Inner Loop (Hyperparameter Tuning)**

* + Prior to each inner loop iteration, a predefined set of hyperparameter combinations is selected. These combinations are generated through grid search.
  + Within the nine folds reserved from the outer loop (i.e., the training + validation set), further cross-validation is performed for hyperparameter optimization.
  + During each inner loop iteration:

One fold is used as the validation set.

The remaining eight folds are used as the training set.

* + The inner loop iterates across all possible partitions of these nine folds, computing performance metrics (e.g., RMSE, R²) for each hyperparameter combination.
  + For each hyperparameter configuration, the performance metric is averaged over the nine iterations to evaluate model effectiveness.

**Model Retraining and Testing After Hyperparameter Selection**

* + After evaluating all hyperparameter combinations via grid search, the optimal set of hyperparameters is selected based on the highest average performance in the inner loop.
  + Using this optimal configuration, the model is retrained on the full nine-fold training + validation set from the current outer loop iteration.
  + The retrained model is then evaluated on the corresponding outer-loop test set (i.e., the held-out fold for that iteration).
  + The resulting performance metric is recorded for final assessment.

**Final Results and Model Deployment**

* + Once the outer loop completes all 10 iterations, a total of 10 performance metrics are obtained — one for each test fold.
  + The model achieving the best overall performance across these iterations is selected for deployment.
  + This nested cross-validation approach ensures robustness in both hyperparameter selection and generalization performance estimation, minimizing the risk of overfitting and information leakage during the model development pipeline.
  1. Baseline Models
     1. Random Forest (RF)

**Model Overview:** Random Forest is an ensemble learning method that constructs multiple decision trees and aggregates their predictions to improve accuracy and robustness.

**Parameter Optimization:** Nested cross-validation is employed where the outer loop evaluates model generalization performance, while the inner loop optimizes hyperparameters (e.g., number of trees, maximum depth, minimum samples split).

**Evaluation Metrics:** Includes Accuracy, AUC, F1 Score, etc., depending on the task type (classification/regression).

* + 1. Multi-Layer Perceptron (MLP)

**Model Overview:** MLP is a feedforward neural network consisting of one or more hidden layers capable of modeling complex non-linear relationships.

**Parameter Optimization:** Also uses nested cross-validation with inner-loop optimization focusing on parameters such as the number of hidden layers, neurons per layer, activation functions, learning rate, regularization coefficients, etc.

**Training Details:** Supports common deep learning techniques like Early Stopping and Batch Normalization to enhance model performance.

**Evaluation and Visualization:** Provides visualization of training processes and outcomes, supporting output and comparison of various performance metrics.

* 1. KAN Model
     1. KAN Overview

**Kolmogorov–Arnold Networks (KAN)** constitute a groundbreaking machine learning network architecture that breaks away from the connectionist approach of Multi-Layer Perceptrons (MLPs). KANs can be conceptualized as language models tailored for AI applications in scientific domains, wherein the "language" of science is defined by mathematical functions. By being constructed from interpretable functions, KANs provide significantly greater transparency and interpretability relative to traditional MLP-based neural networks, which are frequently perceived as opaque black-box systems. Consequently, KANs exhibit superior suitability for tackling scientific problems where understanding and explaining the underlying mechanisms are paramount.

* + 1. Parameter Optimization

Two-phase optimization strategy:

1. Architecture Search:

• Grid search for optimal network width/depth combinations

• Cross-validation for spline parameters (grid count k∈[3,7], order grid∈[5,20])

• Adaptive determination of multiplicative node order (mult\_arity)

1. Training Optimization:

• Dynamic learning rate adjustment (ReduceLROnPlateau)

• Early stopping (patience=15)

• Mixed-precision training support

* + 1. Feature Importance Analysis

Multi-level interpretability capabilities:

1. Quantitative Analysis:

• Gradient-based global importance scoring

• Layer-wise feature contribution decomposition

1. Symbolic Parsing:

• Automated symbolic regression for human-readable formulas

1. Interaction Detection:

• Identifies feature interactions via multiplicative nodes

• Visual heatmaps for cross-feature effects

* + 1. Visualization Capabilities

Comprehensive visualization toolkit:

1. Network Architecture Visualization:

• Interactive display of layer activation patterns

• Animated spline basis function adjustment process

1. Analytical Visualization:

• Radar/bar charts for feature importance

• Rendered mathematical expressions of symbolic formulas

• Prediction comparison scatter plots

1. Training Monitoring:

• Real-time loss curves

• Parameter distribution histograms

• Gradient flow visualization

This design maintains the powerful fitting capability of neural networks while breaking through the interpretability limitations of traditional black-box models, providing a new technical option for critical domains requiring reliability and explainability.

1. User Interface Design
   1. Page Overview

|  |  |
| --- | --- |
| Page name | Functions |
| Login / Register Page | User login and registration interface, supports login by email or username, supports email registration, and verification by verification code. |
| Grid Page | The grid data display page uses a heat map to render the surface temperature and night light data of Shanghai. |
| Analyze Page | Key Factor Analysis Page: By uploading Excel data, analyze the key factors of the light and heat data in the independent variables |
| Compare Page | Model comparison page showing the prediction ability comparison between the kan model and other benchmark models |
| Predict Page | Photothermal data prediction page, upload independent variable data, and predict the corresponding photothermal data |

* 1. Page functions and interaction details
     1. Login / Register Page

• User login module (email/username + password)

• User registration module (email + verification code + password setting)

• Verification code acquisition and countdown function

• Automatically jump to the home page after login, Token is stored in localStorage

* + 1. Grid Page(Visual heat map)

• Map selection (switch surface temperature/night light)

• ECharts heat map rendering

• Legend and zoom control

• Layer switching and color range control

* + 1. Analyze Page(Key Factor Analysis)

• Excel data upload (based on el-upload)

• Select photothermal variables

• One-click analysis button

• Analysis result chart (bar chart, factor ranking)

* + 1. Compare Page(Model comparison)

• Model selection (KAN, RF, MLP)

• Model prediction result graph (line graph)

• Model error indicator display (MAE, R²)

• Compare different model results in the same graph

* + 1. Predict Page(Photothermal prediction)

• Upload input data (Excel table format)

• One-click prediction and rendering of result graph (line graph)

• Support prediction result download (CSV/Excel)

1. Interface Specification
   1. API Design
      1. API Design Principles

RESTful Style: All services follow RESTful API design principles. Resources are manipulated via HTTP methods (GET/POST/PUT/DELETE), ensuring clear interface semantics.

Layered Architecture: The system adopts a microservices architecture. APIs are uniformly routed and aggregated through an API Gateway, while backend services (Authentication, Data, Prediction, Comparison, Analysis) are independently deployed for clear decoupling.

Consistency & Standardization:

All endpoints use the prefix /api/v1/ followed by the service name (e.g., /auth, /data, /predict).

Uniform naming conventions for endpoints, parameters, and response structures.

Standardized Responses: All responses use a unified JSON format:

|  |
| --- |
| {  "code": status.HTTP\_200\_OK, // HTTP status code  "message": "Success", // Human-readable message  "data": {} // Response payload  } |

Error Handling:

Standard HTTP status codes (4xx/5xx) with detailed error messages in the message field.

|  |
| --- |
| HTTPException(  status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR, // HTTP status code  detail="预测失败或未找到指定的因变量列" // detailed error messages  ) |

Automated Documentation: All services integrate OpenAPI/Swagger for interactive documentation, simplifying development and testing.

* + 1. Authentication

Unified Authentication:

JWT (JSON Web Token) secures all authenticated endpoints.

auth\_service handles user registration, login, token issuance, and validation.

OAuth2 Compliance:

Implements OAuth2 Password Grant.

Clients obtain access\_token via POST /api/v1/auth/login.

Subsequent requests include header: Authorization: Bearer <token>.

Inter-Service Security:

Microservices communicate securely using shared secrets and JWT validation.

CORS Support:

All services configured with Cross-Origin Resource Sharing (CORS) for specified frontend domains.

Supports credentials (cookies) and custom headers.

* + 1. Core Interfaces

API Gateway (Unified Entry)

|  |  |
| --- | --- |
| **Endpoint** | **Description** |
| GET /api/v1/health | Service health check. |
| GET /api/v1/ | API Gateway metadata. |

Authentication Service (auth\_service)

|  |  |
| --- | --- |
| **Endpoint** | **Description** |
| POST /api/v1/auth/register | User registration |
| POST /api/v1/auth/login | User login; returns JWT. |
| POST /api/v1/auth/reset\_password | Password reset. |
| POST /api/v1/auth/register/send-code | Get code to register. |
| POST /api/v1/auth/reset-password/send-code | Get code to reset password. |

Data Service (data\_service)

|  |  |
| --- | --- |
| **Endpoint** | **Description** |
| GET /api/v1/data/get\_data | Fetches raster data for target variables. |
| **target:**   * nighttime\_ * lst\_day\_c * lst\_night\_c | |

Prediction Service (predict\_service)

|  |  |
| --- | --- |
| **Endpoint** | **Description** |
| POST /api/v1/predict/dependent\_predict | Submits data for prediction for target variables using KAN Model. |
| **target:**   * nighttime\_ * lst\_day\_c * lst\_night\_c | |

Comparison Service (compare\_service)

|  |  |
| --- | --- |
| **Endpoint** | **Description** |
| POST /api/v1/compare/model\_compare | Compares results for target variables from multiple prediction models. |
| **target:**   * nighttime\_ * lst\_day\_c * lst\_night\_c | |

Analysis Service (analyze\_service)

|  |  |
| --- | --- |
| Endpoint | Description |
| POST /api/v1/analyze/dependent\_feature\_analyze | Returns feature importance scores for target variables for input data. |
| **target:**   * nighttime\_ * lst\_day\_c * lst\_night\_c | |

* + 1. Detailed Design
       1. auth\_service (Authentication Service)
* User Login

Method: POST

Endpoint: /api/v1/auth/login

Content-Type: application/json

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "username": "string",  "password": "string"  } | {  "code": 200,  "message": "success",  "data": {  "token": "string",  "expire": 0,  "role": "string",  "username": "string"  }  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* Send Registration Code

Method: POST

Endpoint: /api/v1/auth/register/send-code

Content-Type: application/json

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "email": "user@example.com"  } | {  "code": 200,  "message": "验证码已发送至邮箱，有效期为10分钟",  "data": null  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* User Registration

Method: POST

Endpoint: /api/v1/auth/register

Content-Type: application/json

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "email": "user@example.com",  "username": "string",  "password": "string",  "verification\_code": "string"  } | {  "code": 200,  "message": "success",  "data": {  "id": user.id,  "username": "string",  "email": user.email,  "is\_active": true,  "created\_at": DateTime  }  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* Send Password Reset Code

Method: POST

Endpoint: /api/v1/auth/reset-password/send-code

Content-Type: application/json

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "email": "user@example.com"  } | {  "code": 200,  "message": "验证码已发送至邮箱，有效期为10分钟",  "data": null  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* Reset Password

Method: POST

Endpoint: /api/v1/auth/reset-password

Content-Type: application/json

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "email": "user@example.com",  "new\_password": "string",  "verification\_code": "string"  } | {  "code": 200,  "message": "密码重置成功，请使用新密码登录",  "data": null  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* + - 1. compare\_service (Model Comparison Service)
* Model Comparison

Method: POST

Endpoint: /api/v1/compare/model\_compare

Content-Type: multipart/form-data

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "file": File,  "dependent\_name": "string"  } | {  "code": 200,  "message": "success",  "data": [  {  "dependent\_name": "string",  "true\_values": [  0  ],  "mlp\_predictions": [  0  ],  "rf\_predictions": [  0  ],  "kan\_predictions": [  0  ],  "mse\_mlp": 0,  "r2\_mlp": 0,  "mse\_rf": 0,  "r2\_rf": 0,  "mse\_kan": 0,  "r2\_kan": 0  }  ]  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* + - 1. data\_service (Grid Data Service)
* Get Data

Method: GET

Endpoint: /api/v1/data/get\_data

Content-Type: application/json

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "data": "string"  } | {  "code": 200,  "message": "success",  "data": [  {  "longitude": 0,  "latitude": 0,  "value": 0  }  ]  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* + - 1. predict\_service (Prediction Service)
* Dependent Variable Prediction

Method: POST

Endpoint: /api/v1/predict/dependent\_predict

Content-Type: multipart/form-data

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "file": File,  "dependent\_name": "string"  } | {  "code": 200,  "message": "success",  "data": [  0  ]  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* + - 1. analyze\_service (Key Feature Analysis Service)
* Dependent Feature Analysis

Method: POST

Endpoint: /api/v1/analyze/dependent\_feature\_analyze

Content-Type: multipart/form-data

|  |  |  |
| --- | --- | --- |
| Request Data Format | Response Data Format | Error Responses |
| {  "file": File,  "dependent\_name": "string"  } | {  "code": 200,  "message": "success",  "data": [  {  "feature\_name": "string",  "feature\_value": 0  }  ]  } | {  "detail": [  {  "loc": [  "string",  0  ],  "msg": "string",  "type": "string"  }  ]  } |

* 1. External Interfaces
     1. Notification Services (Email Verification Notifications)

**Registration Verification Notification:** When a user registers, the system sends a registration verification code to the user's email. The user needs to enter the verification code to complete the registration process.

**Password Reset Verification Notification:** When a user forgets their password, the system sends a password reset verification code to the user's email. The user can reset the password after entering the verification code.

Generating a random 6-digit verification code.

Storing the verification code, type (registration/reset), and expiration time (10 minutes) in memory.

Sending the email via the SMTP protocol, supporting SSL encryption. The email content is automatically generated based on its purpose (registration/reset) and contains the verification code and validity information.

* 1. User Interface to Backend Integration
     1. Integration Patterns

Data Fetching:

* + Utilization of RESTful API calls for CRUD operations to interact with backend services.
  + Structured data stored in backend storage systems can be retrieved via dedicated endpoints.
  + Machine learning models are loaded and executed on-demand via model-serving interfaces.
  + File-based inputs are submitted using multipart/form-data format through frontend components such as file uploaders.

Authentication Flow:

* + Secure submission of login credentials via form-based authentication.
  + Upon successful authentication, a JWT token is issued and stored in browser storage.
  + This token must be included in the Authorization header for all subsequent authenticated requests.
  + An automatic token refresh mechanism is implemented to handle session prolongation before token expiration.

Error and Exception Handling

* + All backend interfaces return standard HTTP status codes and detailed error messages. The frontend should provide user-friendly prompts based on these status codes and message fields.
  + Frontend displays user-friendly messages based on HTTP status codes and error descriptions.

State Management

* + Application state is managed through a centralized store.
  + Frequently accessed data is cached with time-based invalidation policies to ensure freshness.
  + Optimistic UI updates are performed to enhance perceived performance, with rollback capabilities upon failure.
    1. Interaction Details

**Unified Entry Point:** All frontend requests pass through an API Gateway (default port 8000), which routes them to the appropriate microservice based on the request path.

**Cross-Origin Resource Sharing (CORS):** The backend is configured to allow specified frontend domains to perform cross-origin resource sharing, including support for cookies and custom headers, ensuring seamless communication during separate frontend and backend deployments.

**Verification Code Mechanism:** During registration and password recovery, the frontend initiates the verification code sending process. Once users receive the code via email, they input it into the frontend and submit it along with other information for validation.

**Session Expiry Management:** If the backend returns 401/403 status codes indicating expired or invalid sessions, the frontend should automatically redirect the user to the login page or prompt for re-authentication.

* + 1. Typical Interaction Flows

User Registration & Login:

* + Users enter their email address and click "Get Verification Code". The system sends a verification code to the user's email.
  + After entering the verification code, username, and password, users complete registration by invoking POST /api/v1/auth/register.
  + For login, users submit their credentials via POST /api/v1/auth/login, receive a JWT token, and use it for future authenticated interactions.

Data Query and Model Execution:

* + Data retrieval from structured files is achieved via GET /api/v1/data/get\_data?data=xxx.
  + Model execution involves uploading structured files (e.g., Excel, CSV) to endpoints such as:

POST /api/v1/predict/dependent\_predict

POST /api/v1/compare/model\_compare

POST /api/v1/analyze/dependent\_feature\_analyze

Result Presentation:

* + The backend returns structured JSON responses, which the frontend parses and renders using visualization components such as tables, charts, maps, etc.

1. Non-Functional Design
   1. Performance Requirements

|  |  |
| --- | --- |
| Indicators | Descriptions |
| Response Time | System page loading response time ≤ 3 seconds; data prediction result response time ≤ 10 seconds; key factor analysis time ≤ 30 seconds |
| Data upload restrictions | The size of a single uploaded data file is ≤ 10 MB, and supports common Excel formats |
| Chart rendering efficiency | Visualization components such as heat maps and line charts are rendered within 2 seconds |

* 1. Security Requirements

|  |  |
| --- | --- |
| Indicators | Descriptions |
| Login verification | Login and registration require email verification and verification code verification to prevent brute force cracking |
| Data storage security | User uploaded data is only used for analysis and prediction and is not stored |
| Interface Security | All interfaces implement authentication mechanism based on FastAPI |

* 1. Usability Requirements

|  |  |
| --- | --- |
| Indicators | Descriptions |
| User-friendly interface | Simple UI, clear navigation, and interactive charts make it suitable for researchers or planners to quickly understand data. |
| Ease of operation | Provide auxiliary information such as operation prompts, button labels, upload format instructions, etc. |
| Error message | Improved form validation and exception handling, users can get clear feedback |

* 1. Maintainability Requirements

|  |  |
| --- | --- |
| Indicators | Descriptions |
| Modular design | The front-end and back-end modules are decoupled, making it easy to upgrade, test and debug independently. |
| Code Comment Standards | All functions and components provide standard comments, and both the front-end and back-end follow the naming conventions |
| Interface documentation | The backend generates interface documents to facilitate front-end and back-end collaboration |

* 1. Compatibility Requirements

|  |  |
| --- | --- |
| Indicators | Descriptions |
| Browser compatibility | Supports mainstream browsers such as Chrome, Edge, Firefox, etc., with compatibility ≥ 95% |
| Front-end and back-end interface standards | Use JSON format for data interaction, stable interface, and support cross-platform calls |