ISSN: 2688-8645

Canteen Recommendation System Based on SpringBoot and SpringAI

Ye Sun, Guixiu Liu, Yuanyuan Wang, Jia Liu, Wangya Huang

School of Computer and Artificial Intelligence, Beijing Wuzi University, Beijing 101149, China

Abstract

This paper presents the design and implementation of a university canteen personalized recommendation system based on SpringBoot and SpringAI. The system adopts a front-end/back-end separation architecture. The backend integrates SpringBoot, MyBatis and a MySQL database, while the frontend is built with Bootstrap to provide a responsive interface. By incorporating large language model capabilities via the SpringAI module, the system supports intelligent dish recommendations and conversational management. Through user profiling, dynamic recommendation algorithms, and multi-canteen switching functionality, the system effectively enhances the intelligence and user experience of campus catering services. This paper describes the system architecture, database modeling, AI integration strategy and core implementation in detail, offering a feasible technical path and practical reference for developing smart catering systems in universities.

Keywords

SpringBoot; SpringAI; Large Language Model; Recommendation System; Canteen Management; Intelligent Dialogue.

1. Introduction

SpringBoot is an open-source project within the Spring ecosystem designed to simplify the initial setup and development of Spring applications. Its 'Convention over Configuration' philosophy greatly reduces the configuration burden for developers, enabling teams to rapidly build production-grade Java applications. SpringBoot provides abundant starter dependencies and automatic configuration mechanisms, facilitating seamless integration with various databases, middleware and third-party services. [1]

SpringAI is a core module in the Spring ecosystem targeting artificial intelligence applications, offering a unified API for integrating large language models (LLMs) and supporting advanced capabilities such as dialogue management, intelligent reasoning and task orchestration. Through SpringAI, developers can readily invoke AI features to implement context-aware conversational systems, dynamic task decomposition and result aggregation, significantly improving system intelligence.

This paper uses a university canteen personalized recommendation system as an example to demonstrate the design and implementation of a web system based on SpringBoot and SpringAI. The system employs a front-end/back-end separated architecture, with a Bootstrap-based responsive frontend and a SpringBoot backend that integrates SpringAI, MyBatis and MySQL to implement user profiling, dynamic recommendation generation, multi-canteen switching and intelligent dialogue. The system is validated and evaluated in real campus scenarios.

ISSN: 2688-8645

2. System Design Goals

2.1. Canteen Dish Management and Data Collection

(1) Dish information entry. The system supports administrators or partnered canteen staff to input basic dish information, including dish name, price, affiliated canteen, serving window, nutritional information, and taste tags (e.g., spicy, sweet, vegetarian).

(2) Dynamic menu updates. The system supports real-time updates to daily menus to ensure students receive accurate dish information and to avoid recommendation errors due to menu changes.

(3) Data standardization. The system processes dish data into structured formats and establishes a unified tagging system to provide the data foundation for intelligent recommendations. [2]

2.2. User Profiling and Preference Analysis

(1) User registration and preference settings. Student users can set personal dietary preferences (e.g., allergies, taste preferences, health requirements) through the app; the system constructs an initial user profile accordingly.

(2) Behavioral data collection. The system records user behaviors such as browsing, favoriting, and rating, dynamically updating user profiles to improve recommendation accuracy.

(3) Clustering and association analysis. Based on user behavior data, the system applies clustering analysis and association rule mining to identify user group characteristics and dish preference patterns.

2.3. Intelligent Recommendation and Dynamic Switching

(1) Personalized dish recommendation. The system generates real-time personalized dish recommendation lists based on user profiles, supporting multi-dimensional filtering by taste, health, scenario, etc.

(2) Dynamic canteen switching. Users can dynamically adjust the recommendation source via the 'switch canteen' function; the system responds and updates recommendations in real time.

(3) Recommendation algorithm optimization. The system integrates collaborative filtering and content-based filtering algorithms and supports real-time optimization mechanisms based on user feedback.

2.4. User Interaction and Feedback Mechanism

(1) Minimalist interaction design. Users can obtain recommendations by answering only 3–5 simple questions, lowering the usage barrier and improving efficiency.

(2) Real-time evaluation function. Users can rate and comment on recommended dishes; the system collects feedback for algorithm iteration and canteen operation optimization.

(3) Intelligent voice assistant. The system supports voice input for queries and recommendations, further enhancing interaction convenience and user experience.

2.5. Data Analysis and Operational Support

(1) Consumption behavior analysis. The system compiles statistical analyses of user dining preferences and consumption frequency to provide operational recommendations for canteens.

(2) Dish popularity rankings. Periodically generate 'Top 5 Student Favorites' lists to assist canteens in adjusting dish supply structures.

(3) Health dietary reports. Generate personalized health suggestions based on user dietary data to support campus health education promotion.

ISSN: 2688-8645

3. System Design

3.1. Development Environment and Technology Stack

The system aims to build an efficient and scalable catering management and online ordering platform. The backend technology choices consider system performance, maintainability and collaboration with the frontend. Key development environment and technology stack include:

- Backend framework: Spring Boot 3.3, providing a lightweight and modular development environment and supporting rapid deployment and microservice expansion;
- Build tools: Maven 3.6 and mvnd (Maven Daemon) 1.0.2 for dependency management and automated builds;
- JDK version: 21.0.8 to support modern language features and high-concurrency handling;
- SpringAI: 1.0.0 to simplify the development of AI-enabled applications;
- Database: MySQL 8.0.33 (InnoDB engine) supporting transactions and foreign-key constraints;
- Persistence framework: MyBatis 1.3.1 combined with MyBatis-Plus 2.1.5 for efficient data access and ORM mapping.

This technology stack balances stability, scalability and development efficiency, meeting the rapid iteration needs of small-to-medium internet applications. [3]

3.2. Database Design

3.2.1. Core Data Tables

The database design includes core modules for user management, dish management, order management and AI conversation storage. Key table functions are summarized below:

User management: admins (system administrators), students (student/user registration), merchants (canteen/shop information and verification).

Dish management: dish_category (dish classification), dish_info (basic dish information and inventory).

Order management: orders (order main table), order_details (order line items), cart (shopping cart), payment_records.

User interaction: comments (user reviews and feedback), online_complaints, favorites.

Other: carousel_images (banner management).

3.2.2. AI-related Tables

The AI module stores chat memory to maintain conversational context. Example table stores conversation_id, content, message type (USER, ASSISTANT, SYSTEM, TOOL) and timestamp to support multi-session isolation and message ordering.

3.2.3. Dish Information Table (caipinxinxi)

SQL schema (translated comments):

```
CREATE TABLE `caipinxinxi` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `dish_code` varchar(50) NOT NULL COMMENT 'Dish code',
  `dish_name` varchar(255) NOT NULL COMMENT 'Dish name',
  `category` int(10) unsigned NOT NULL COMMENT 'Category',
  `dish_image` varchar(255) NOT NULL COMMENT 'Dish image',
  `rating` decimal(18,2) NOT NULL COMMENT 'Rating',
  `price` decimal(18,2) NOT NULL COMMENT 'Price',
  `discount_price` int(11) NOT NULL COMMENT 'Discount price',
  `sales` int(11) NOT NULL COMMENT 'Sales',
```

ISSN: 2688-8645

```
  `stock` int(11) NOT NULL COMMENT 'Inventory',
  `dish_details` longtext NOT NULL COMMENT 'Dish details',
  `publisher` varchar(50) NOT NULL COMMENT 'Publisher',
  `addtime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Added time',
  PRIMARY KEY (`id`),
  KEY `caipinxinxi_category_index` (`category`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.2.4. Order Table (dingdan)

```
CREATE TABLE `dingdan` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `order_code` varchar(50) NOT NULL COMMENT 'Order code',
  `buyer` varchar(50) NOT NULL COMMENT 'Buyer',
  `order_amount` decimal(18,2) NOT NULL COMMENT 'Order amount',
  `order_details` text NOT NULL COMMENT 'Order details',
  `note` text NOT NULL COMMENT 'Remarks',
  `is_paid` varchar(10) NOT NULL DEFAULT 'No' COMMENT 'Payment status',
  `addtime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Added time',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.2.5. AI Conversation Memory Table

```
CREATE TABLE IF NOT EXISTS SPRING_AI_CHAT_MEMORY (
    conversation_id VARCHAR(36) NOT NULL,
        content TEXT NOT NULL,
        type VARCHAR(10) NOT NULL,
        `timestamp` TIMESTAMP NOT NULL,
        CONSTRAINT TYPE_CHECK CHECK (type IN ('USER', 'ASSISTANT', 'SYSTEM', 'TOOL'))
    );

    CREATE INDEX SPRING_AI_CHAT_MEMORY_CONVERSATION_ID_TIMESTAMP_IDX
    ON SPRING_AI_CHAT_MEMORY(conversation_id, `timestamp`);
```

3.3. Backend File Design

3.3.1. Three-tier Architecture

The backend adopts the classic three-tier architecture pattern including:

(1) Presentation layer (Controller): manages HTTP requests and responses and provides RESTful APIs;

(2) Business logic layer (Service): encapsulates core business logic and ensures transactional consistency;

(3) Data access layer (DAO/Mapper): interacts with the database via MyBatis.

This architecture decouples functional modules, facilitating maintenance, extension and future microservice migration.

3.3.2. AI Architecture and Core Code Examples

Figure 1: AI architecture design (retain original image if present).

3.3.2.1 Catering part - Base Service Class (ServiceBase.java):

```
abstract public class ServiceBase<E> {
```

ISSN: 2688-8645

```
    abstract protected Mapper<E> getDao();

    public List<E> select() {
        return getDao().select(null);
    }

    public E find(Object id) {
        return getDao().selectByPrimaryKey(id);
    }

    public List<E> selectPage(E obj, int page, int pageSize) {
        PageHelper.startPage(page, pageSize, true);
        List<E> list = select(obj);
        PageInfo<E> pageInfo = new PageInfo<E>(list);
        return list;
    }

    public int insert(E y) {
        return getDao().insertSelective(y);
    }

    public int update(E y) {
        return getDao().updateByPrimaryKeySelective(y);
    }

    public int delete(Object id) {
        return getDao().deleteByPrimaryKey(id);
    }
}
```

Base Controller (BaseController.java):

```
abstract public class BaseController {
    @Autowired
    protected HttpServletRequest request;
    @Autowired
    protected HttpServletResponse response;
    @Autowired
    protected HttpSession session;

    protected String showError(String message) {
        return showMessage(message, 1, "javascript:history(-1);", 2250);
    }

    protected String showSuccess(String message) {
        return showMessage(message, 0, request.getHeader("referer"), 2250);
```

ISSN: 2688-8645

```
    }

  protected boolean checkLogin() {
      return session.getAttribute("username") != null;
    }
}
```

Example business controller (CaipinxinxiController.java):

```
@Controller
public class CaipinxinxiController extends BaseController {
  @Autowired
  private CaipinxinxiService service;

  @RequestMapping("/caipinxinxi_list")
    public String list() {
      if(!checkLogin()) return showError("Not logged in", "./login.do");

    Example example = new Example(Caipinxinxi.class);
    Example.Criteria criteria = example.createCriteria();
    criteria.andCondition(getWhere());
    example.orderBy("id").desc();

      List<Caipinxinxi> list = service.selectPageExample(example, page, 12);
      request.setAttribute("list", list);
      return "caipinxinxi_list";
    }
}
```

3.3.2.2 AI Part - ChatMemoryService Implementation

```
@Service
public class ChatMemoryService {

  @Autowired
  private ChatModel chatModel;

  @Autowired
  private JdbcChatMemoryRepository chatMemoryRepository;

  private ChatMemory chatMemory;

  @PostConstruct
  public void init() {
    this.chatMemory = MessageWindowChatMemory.builder()
        .chatMemoryRepository(chatMemoryRepository)
        .maxMessages(20)  // limit message window size
        .build();
  }
```

ISSN: 2688-8645

```
    public String call(String message, String conversationId) {
        // 1. create user message
        UserMessage userMessage = new UserMessage(message);

        // 2. store user message to memory
        this.chatMemory.add(conversationId, userMessage);

        // 3. get conversation history from memory
        List<Message> messages = chatMemory.get(conversationId);

        // 4. call ChatModel to generate response
        ChatResponse response = chatModel.call(new Prompt(messages));

        // 5. store AI response to memory
        chatMemory.add(conversationId, response.getResult().getOutput());

        return response.getResult().getOutput().getText();
    }
}
```

3.3.3. Configuration Example

```
# application.properties
spring.application.name=spring-ai-chat-memory-jdbc
server.port=8083

# JDBC Memory Repository configuration
spring.ai.chat.memory.repository.jdbc.initialize-schema=always
spring.ai.chat.memory.repository.jdbc.schema=classpath:schema-@@platform@@.sql
spring.ai.chat.memory.repository.jdbc.platform=mysql

# MySQL datasource configuration
spring.datasource.url=jdbc:mysql://localhost:3306/spring_ai_chat_memory?useUnicode=true
&characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=${spring.datasource.password}
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

3.3.4. System Design Features

(1) Highly encapsulated CRUD operations: Implement common CRUD via a ServiceBase base class to reduce code duplication.

(2) Unified exception handling: BaseController provides unified message prompts; front-end/back-end separated exception handling.

(3) Flexible query conditions: Use MyBatis Example to build dynamic queries supporting pagination, sorting and filtering.

(4) Access control: Session-based simple permission verification with a three-tier permission model (admin, merchant, student).

(5) Front-end/back-end collaboration: JSP templates, RESTful URLs and AJAX support.

ISSN: 2688-8645

(6) Simple interface for interacting with DeepSeek.

(7) Robust Chat Memory mechanism persisting conversation history in the database.

4. Conclusion

Based on the Spring Boot + MyBatis technology stack and a three-tier architecture, the system achieves modular decoupling. Following third normal form database design, combined with index optimization and transaction management, ensures responsiveness under high concurrency. Unified CRUD, permission management and dynamic querying reduce iteration costs and meet current online catering management requirements while laying the groundwork for future distributed or microservice architectures. Additionally, the integration of Spring AI Chat Memory with a JDBC-backed storage implements conversational memory persistence to maintain context across interactions, improving user experience. The JDBC approach provides data persistence and scalability. This project demonstrates technical innovations in both catering management and intelligent dialogue, offering a valuable reference for future extensions.

Acknowledgments

I would like to thank my group members for their invaluable support and collaboration.

References

[1] Y. Liu. Design of a Homework Management System Based on SpringBoot and MyBatis. Computer Programming Techniques & Maintenance, 2025(04):86-88.

[2] J. Guo. Design and Implementation of a Student Moral Evaluation System Based on SpringBoot+MyBatis+Vue. Modern Information Technology, 2023, 7(01):18-22. DOI:10.19850/j.cnki. 2096-4706.2023.01.004.

[3] K. Yang. In-depth Introduction to Spring Boot 3.x. People's Posts and Telecommunications Press, 2024.