Project 2's Report of CS307 - Principles of Database Systems (2024 Spring)

Contributors: ZERHOUNI KHAL Jaouhara (12211456) & HOK Layheng (12210736)
Instructor: Dr. MA Yuxin
Lab Session: Tuesday (5-6)

TABLE OF CONTENTS

I. CONTRIBUTION

Members	Tasks	Ratio
ZERHOUNI KHAL Jaouhara	- Import new data (price) - CRUD on station data - CRUD on the line detail data (relationship between stations and lines) - Search for the n-th station's details that come before or after a specified station on a line - Design a comprehensive system for stations buses and landmarks integration - UI/UX design - Report	50%
HOK Layheng	- Set up project and dependencies with Maven - Set up triggers and procedures - Build API with Spring Boot - CRUD on the line data - Boarding and exiting functionalities - View information about ongoing rides passengers and cards - Utilize the status of stations and different ride classes - Filter ride records with multi-parameter inputs and utilize pagination to handle large ride results	50%

II. BASIC REQUIREMENTS: API SPECIFICATIONS

1. STATION

- **Purpose**: Manage station record.
- **Use**: Display all stations, create new stations, update existing stations' details, or remove stations from the database.
- **API**:

- **Show Station List Page**: Retrieves a list of all stations and displays them on the Stations list page.

  Endpoint: `/stations`
  Method: `GET`
  Parameters: `model` (Model :interface, used to pass data from the controller to the view)
  Request Example: GET /stations
  Response Example (HTML Page):

  ## Stations

  | ← | Add a Station | | | | |

English Name	Chinese Name	District	Intro	Status	Actions
Ailian	爱联	龙岗区	爱联站（Ailian Station）是中国广东省深圳市境内地铁车站，位于中国广东省深圳市龙岗区境内，是深圳市地铁集团有限公司管理运营的地铁车站，也是深圳地铁3号线中间站。爱联站于2010年12月28日投用运营，通行深圳地铁3号线。据2023年3月深圳地铁官网显示，爱联站共开通4个出入口，工作日运营时间为06:14至次日00:03，休息日运营时间为06:17至次日00:02，节假日运营时间为06:17至次日	Operational	Update Remove View Buses View Landmarks
Airport	机场	宝安区	机场站是深圳地铁11号线的一座车站，位于中国广东省深圳市深圳宝安国际机场航站楼及交通中心北侧为，在T3航站楼地下，在航站楼南侧为机场交通中心，交通中心南侧规划为景观空地。乘客可出站通过通道换乘穗深城际铁路深圳机场站。机场站在11号线上编号为9，该站为地下站。机场站于2016年6月28日正式启用。机场站是深圳地铁11号线的一座车站，位于中国广东省深圳市深圳宝安国际机场航	Operational	Update Remove View Buses View Landmarks
Airport East	机场东	宝安区	机场东站是深大城际深圳段的一个站点，位于深圳宝安国际机场A、B航站楼东侧机场四路下方。机场东片区将规划为大型交通枢纽，枢纽内由东向西依次包含既有深圳地铁1号线、深圳地铁12号线、规划地铁20/26号线、深大城际、深茂铁路、APM陆侧、A/B航站楼等主要交通设施。机场东站是深大城际深圳段的一个站点，位于深圳宝安国际机场A、B航站楼东侧机场四路下方。机场东片区将规划为大型	Operational	Update Remove View Buses View Landmarks
Airport North	机场北	宝安区	机场北站（Airport North Station），位于中国广东省深圳市宝安区境内，是深圳市地铁集团有限公司管理运营的地铁车站，也是深圳地铁11号线与深圳地铁20号线的换乘站。机场北站于2016年6月28日投用运营，通行深圳地铁11号线；于2021年12月28日通行深圳地铁20号线。据2022年8月深圳地铁官网显示，机场北站共开通1个出入口，工作日运营时间为06:05-次日00:08。机场北站（Airport North	Operational	Update Remove View Buses View Landmarks

  Errors: None

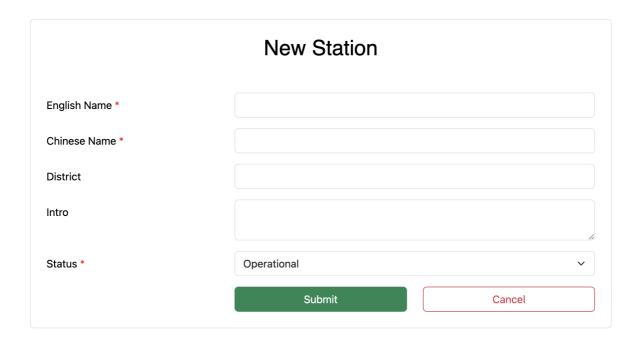- **Show Station Create Page**: Displays the form for creating a new station.

  Endpoint: `/stations/create`
  Method: `GET`
  Parameters: `model` (Model :interface, used to pass data from the controller to the view)
  Request Example: GET /stations/create
  Response Example (HTML Form):

New Station

English Name *

Chinese Name *

District

Intro

Status *   Operational

| Submit | Cancel |

Errors: None

○ **Create Station**: Creates a new station with the provided details.

Endpoint: `/stations/create`
Method: `POST`
Parameters:

- `stationDto` (StationDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `englishName` (string, required), `chineseName` (string, required), `district` (string, optional), `intro` (string, optional), and `status` (string, optional))
- `bindingResult` (BindingResult: interface, used to hold the results of a validation and binding operation for a @ModelAttribute StationDto stationDto, containing errors and validation messages if any occurred during the data binding process)

Request Example:

```
{
  "englishName": "Central Station",
  "chineseName": "中央车站",
  "district": "Nanshan",
  "intro": "中央车站 (Central Station) has been opened to the
public for commuting since June 5th, 2023.",
  "status": "Operational"
}
```

Response Example:

```
HTTP/1.1 302 Found
Location: redirect:/stations
```

Errors:

```
  - 400 Bad Request: Missing or invalid parameters.
  - 409 Conflict: Station with the same name already exists.
```

- **Show Station Update Page**: Displays the form for updating the details of an existing station.

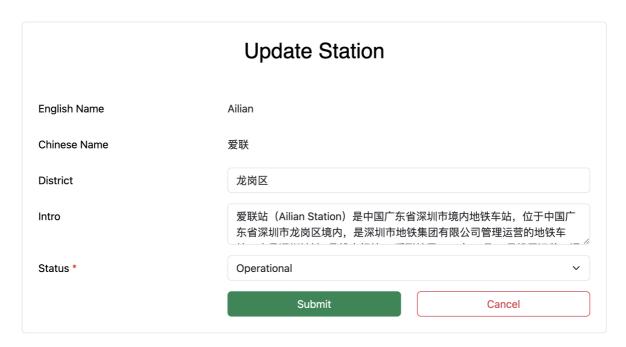 **Endpoint**: `/stations/update`
  Method: `GET`
  Parameters: `englishName` (string, the English name of the station to be updated), `model` (Model)
  Request Example: GET /stations/update?englishName=Ailian
  Response Example (HTML Form):

  Errors:

  ```
    - 404 Not Found: Station not found.
  ```

- **Update Station**: Updates the details of an existing station.

  Endpoint: `/stations/update`
  Method: `POST`
  Parameters:

  - `stationDto` (StationDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `district` (string, optional), `intro` (string, optional), `status` (string, optional))
  - `bindingResult` (BindingResult)

Request Example:

```
{
  "district": "龙岗区",
  "intro": "This is the updated intro.",
  "status": "Closed"
}
```

Response Example:

```
HTTP/1.1 302 Found
Location: redirect:/stations
```

Errors:

```
— 400 Bad Request: Missing or invalid parameters.
```

- ○ **Remove Station**: Removes an existing station.

  Endpoint: /stations/remove
  Method: GET
  Parameters: englishName (string, required, the English name of the station to be removed), model (Model)

  Request Example: GET /stations/remove?englishName=Ailian

  Response Example:

  ```
  Location: redirect:/stations
  ```

  Errors: None

2. LINE

- **Purpose**: Manage line record.
- **Use**: Display all lines, create new lines, update existing lines' details, or remove lines from the database.
- **API**:

  - ○ **Show Line List Page**: Retrieves a list of all lines and displays them on the Lines list page.

    Endpoint: /lines
    Method: GET
    Parameters: model (Model :interface, used to pass data from the controller to the view)

- **Show Line Create Page**: Displays the form for adding a new line.

  Endpoint: `/lines/create`
  Method: `GET`
  Parameters: `model` (Model :interface, used to pass data from the controller to the view)

- **Create Line**: Creates a new line with the provided details.

  Endpoint: `/lines/create`
  Method: `POST`
  Parameters:

  - `lineDto` (LineDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `lineName` (string, required), `startTime` (time, required), `endTime` (time, required), `mileage` (double, optional), `color` (string, optional), `firstOpening` (date, optional), `intro` (string, optional), `url` (string, optional))
  - `bindingResult` (BindingResult)
    Request Example:

    ```
    {
      "lineName": "1号线",
      "startTime": "06:10:00",
      "endTime": "16:10:00",
      "mileage": "50.50",
      "color": "粉色",
      "firstOpening": "2010-01-22",
      "intro": "line 1 is now open!",
      "url": "https://line1isopen"
    }
    ```

- **Update Line**: Updates the details of an existing line.

  Endpoint: `/lines/update`
  Method: `POST`
  Parameters:

  - `lineDto` (LineDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `startTime` (time, required), `endTime` (time, required), `mileage` (double, optional), `color` (string, optional), `firstOpening` (date, optional), `intro` (string, optional), `url` (string, optional))
  - `bindingResult` (BindingResult)

  Request Example:

    ```
    {
      "startTime": "06:20:00",
      "endTime": "23:00:00",
      "mileage": "40,876",
    ```

```
    "color": "绿色",
    "firstOpening": "2004-12-28",
    "intro": "Updated intro.",
    "url": "[Closed](https://baike.baidu.com/item/深圳地铁1号
线/6178769?fromModule=lemma_inlink)"
}
```

- **Remove Line**: Removes an existing line.

  Endpoint: `/lines/remove`
  Method: `GET`
  Parameters: `lineName` (string, required, the line name to be removed), `model` (Model)

3. STATION AND LINE MANAGEMENT

- **Purpose**: Manage the placement and removal of stations on subway lines.
- **Use**: Place one or more stations at a specified location on a line or remove a station from a line.
- **API**:

  - **Show Line/Station List Page**: Retrieves a list of all lines and stations displays them on the Line Details list page.

    Endpoint: `/lineDetails`
    Method: `GET`
    Parameters: `model` (Model :interface, used to pass data from the controller to the view)

  - **Show Station Create Page**: Displays the form for adding one or more new stations.

    Endpoint: `/lineDetails/create`
    Method: `GET`
    Parameters: `model` (Model :interface, used to pass data from the controller to the view)

  - **Create Station**: Creates a new line with the provided details.

    Endpoint: `/lineDetails/create`
    Method: `POST`
    Parameters:

    - `lineDetailDto` (LineDetailDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `lineName` (string, required), `stationName` (string, required), `stationOrder` (integer, required))
    - `bindingResult` (BindingResult)
      Request Example:

      ```
      {
        "lineName": "1号线",
        "statioName": "Luohu",
        "stationorder": "1"
      }
      ```

- **Remove Station**: Removes an existing station.

  Endpoint: `/lineDetails/remove`
  Method: `GET`
  Parameters: `stationName` (string, required, the station name to be removed), `model` (Model)

4. SEARCH STATIONS

- **Purpose**: Retrieve station names based on their position relative to a specific station.
- **Use**: Find the names of the stations that are the n-th station ahead or behind a specific station on a line.
- **API**:

  - **Show Station Search Page**: Displays the form for searching for a specific station.

    Endpoint: `/lineDetails/search`
    Method: `GET`
    Parameters: `model` (Model :interface, used to pass data from the controller to the view)

  - **Search Station**: Finds a station with the provided details.

    Endpoint: `/lineDetails/search`
    Method: `POST`
    Parameters:

    - `lineDetailSearchDto` (LineDetailSearchDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `lineName` (string, required), `stationName` (string, required), `offset` (integer, required))
    - `bindingResult` (BindingResult)
      Request Example:

      ```
      {
        "lineName": "1号线",
        "statioName": "Luohu",
        "offset": "2"
      }
      ```

5. BOARDING FUNCTIONALITY

- **Purpose**: Record boarding information for passengers or cards.
- **Use**: Record the starting station, boarding time, and passenger or card details when they board the subway.
- **API**:

  - **Show Boarding Info List Page**: Retrieves a list of all lines and stations displays them on the Line Details list page.

    Endpoint: `/rides/create`
    Method: `GET`

Parameters: `model` (Model :interface, used to pass data from the controller to the view)

- **Create Ride**: Creates a new line with the provided details.

  Endpoint: `/rides/create`
  Method: `POST`
  Parameters:

  - `rideDto` (RideDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `userNum` (string, required), `startStation` (string, required), `rideClass` (string, required))
  - `bindingResult` (BindingResult)
    Request Example:

  ```
  {
    "userNum": "140121195012160804",
    "statioName": "Taiziwan",
    "rideClass": "Economy"
  }
  ```

6. EXIT FUNCTIONALITY

- **Purpose**: Record exit information for passengers or cards and calculate trip cost.
- **Use**: Record the destination station, exit time, and calculate the price based on `Price.xlsx` when passengers or cards exit the subway.
- **API**:

  - **Show Exit Info List Page**: Retrieves a list of all lines and stations displays them on the Line Details list page.

    Endpoint: `/rides/update`
    Method: `GET`
    Parameters: `model` (Model :interface, used to pass data from the controller to the view)

  - **Create Ride**: Creates a new line with the provided details.

    Endpoint: `/rides/update`
    Method: `POST`
    Parameters:

    - `rideDto` (RideDto: object, a Data Transfer Object used to encapsulate the data for a station including the attributes like `endStation` (string, required))
    - `bindingResult` (BindingResult)
      Request Example:

    ```
    {
      "endStation": "Shenwai Senior Campus"
    }
    ```

7. VIEW CURRENT BOARDED PASSENGERS/CARDS

- **Purpose**: Display information about passengers or cards currently on board.
- **Use**: View all information about passengers or cards who have boarded but have not yet exited at the current time.
- **API**:

  - **Show Ongoing Rides List Page**: Retrieves a list of information about the ongoing rides displays them on the Ride Record list page.

    Endpoint: `/rides/ongoingRides`
    Method: `GET`
    Parameters: `model` (Model :interface, used to pass data from the controller to the view)

III. ADVANCED REQUIREMENTS

- **Database Implementation**:

  - **Requirement**: Complete the project using OpenGauss or MySQL database.
  - **Implementation**: The project utilizes MySQL database to manage and store all necessary data including stations, lines, passenger details, and pricing information.

- **Station Status Management**:

  - **Requirement**: Add and appropriately utilize the status of stations (e.g., under construction, operational, closed).
  - **Implementation**: Extended the station management system to include status indicators. Stations can now have statuses like "under construction", "operational", and "closed." These statuses are stored in the database and can be updated via API endpoints.

- **Business Carriage in the Subway**:

  - **Requirement**: Include functionality for business carriages in the subway.
  - **Implementation**: Added a new feature that designates certain carriages as business carriages alongside a price adjustment (up to 30%) of the original price. Passengers can book seats in these carriages.

- **Integration of Buses and Subways**:

  - **Requirement**: Establish a comprehensive system to integrate buses and subways.
  - **Implementation**: Developed an integrated transport management system that allows for combined queries and ticketing for both buses and subways. Users can view bus details associated with specific subway stations, manage these associations, and plan routes that include both modes of transport.

- **Integration of Landmarks**:

  - **Requirement**: Establish a comprehensive system to integrate landmarks and subways.
  - **Implementation**: Users can view landmark details associated with specific subway stations.

- **Multi-Parameter Search for Ride Records**:

    - **Requirement**: Enable searching ride records based on multiple parameters.
    - **Implementation**: Created a robust search functionality in the API that allows users to search ride records based on various parameters such as subway stations, passengers, periods, and more.

- **View User Details**:

    - **Using the developed system users can view detailed information about passengers and cards.**
        - **Passenger Details**:
            - If a user clicks on a passenger's ID number, they can view detailed passenger information including:
                - National ID
                - Name
                - Gender
                - Phone Number
                - Origin
        - **Card Details**:
            - If a user clicks on a card number, they can view detailed card information including:
                - Code
                - Balance
                - Created Time

- **Package Management**:

    - **Tool Used**: Maven
    - **Implementation**: Maven was used for managing project dependencies, ensuring all required libraries and tools were included and properly configured in the project.

- **Using Sockets or HTTP/RESTful Web**:

    - **Tool Used**: Spring Boot
    - **Implementation**: Developed a backend server using Spring Boot to handle HTTP/RESTful requests. This allows the application to support various API endpoints for managing stations, lines, passenger details, and more.

- **Using Backend Frameworks or ORM Mapping**:

    - **Tool Used**: JPA (Jakarta Persistence API) with Spring Boot
    - **Implementation**: Used JPA for Object-Relational Mapping (ORM) to interact with the MySQL database. This simplifies database operations by allowing developers to work with Java objects instead of raw SQL queries.

- **Big Data Management**:

    - **Requirement**: Implement big data management to efficiently handle large datasets.

- **Implementation**: Utilized pagination to display large data sets in manageable pages. Specifically in the ride table, large datasets are broken into pages with each page fetching 100 rows of data. This approach ensures that the system remains responsive and the data is easily navigable for users.

- **Effective Presentation and Communication**:

  - **Requirement**: Ensure effective presentation and communication of data and functionalities.
  - **Implementation**: Developed a simple, interactive, and responsive web page UI. The design is optimized to work seamlessly across all forms of devices, including mobile, tablet, and desktop. The interface features smooth scrolling and animations, providing a user-friendly and engaging experience for users.

- **Appropriate usage of database functionalities**:

  - **Requirement**: Appropriately utilize database procedures, triggers and indexes.
  - **Implementation**:

    - **Procedures**: Created stored procedures to handle insert, update, and delete operations related to the ride table, ensuring proper synchronization with the ongoing_ride table.

    - **Triggers**: Implemented triggers to automate and enforce rules for insertions, updates, and deletions in the ride table, ensuring consistency with the ongoing_ride table.

    - **Indexes**: Utilized primary keys and unique constraints which implicitly create indexes, optimizing query performance for key operations.

IV. CONCLUSION

In this project, we successfully designed and implemented a comprehensive subway management system that meets both basic and advanced requirements. We utilized a variety of technologies including Spring Boot, MySQL, and JPA, and effectively integrated multiple functionalities such as station and line management, ride recording, and passenger information handling. Our system also incorporates advanced features like business carriage management, integration with buses and landmarks, and robust search capabilities. The project's efficient handling of large datasets, interactive UI/UX design, and adherence to database best practices demonstrate a solid understanding of database systems principles. This project highlights our ability to work collaboratively and deliver a functional, user-friendly application.