OOP Capstone Project Report

A Java app of Vietnam History using scraped data from external sources.

**Group 7**

Vu Duc An 20215174

Nguyen Tieu Phuong 20210692

Nguyen Dac Tam 20210763

Nguyen Tran Nhat Quoc 20210726

Lecturer: Prof. Trinh Tuan Dat

July 2023

Semester 2022.2

Hanoi University of Science and Technology

# Project Roster & Contribution

We start by independently exploring the problem at hand, allowing each of us to bring our unique perspectives and ideas to the table. We then come together to discuss the challenges we face individually, helping one another overcome obstacles and refine our solutions. In the end, we merge our individual results into a cohesive and comprehensive final project.

| Member Name | Student ID | Tasks | Contribution (%) |
| --- | --- | --- | --- |
| Nguyen Tieu Phuong | 20210692 | Final Report & Presentation; UML diagrams | 25 |
| Vu Duc An | 20215174 | Wikipedia crawl, data aggregation | 25 |
| Nguyen Dac Tam | 20210763 | Person Crawl (Google, nguoikesu, Wiki) | 25 |
| Nguyen Tran Nhat Quoc | 20210726 | UI design, Relic and Festival Crawl, Demo video. | 25 |

# Statistics of Data Collected

We deploy a web crawler to gather data from a diverse range of websites, including notable sources like Wikipedia, Nguoikesu, and Google. Initially, the data we amass is vast in quantity. However, through meticulous cleaning and filtering processes, only a number of usable and refined data remained.

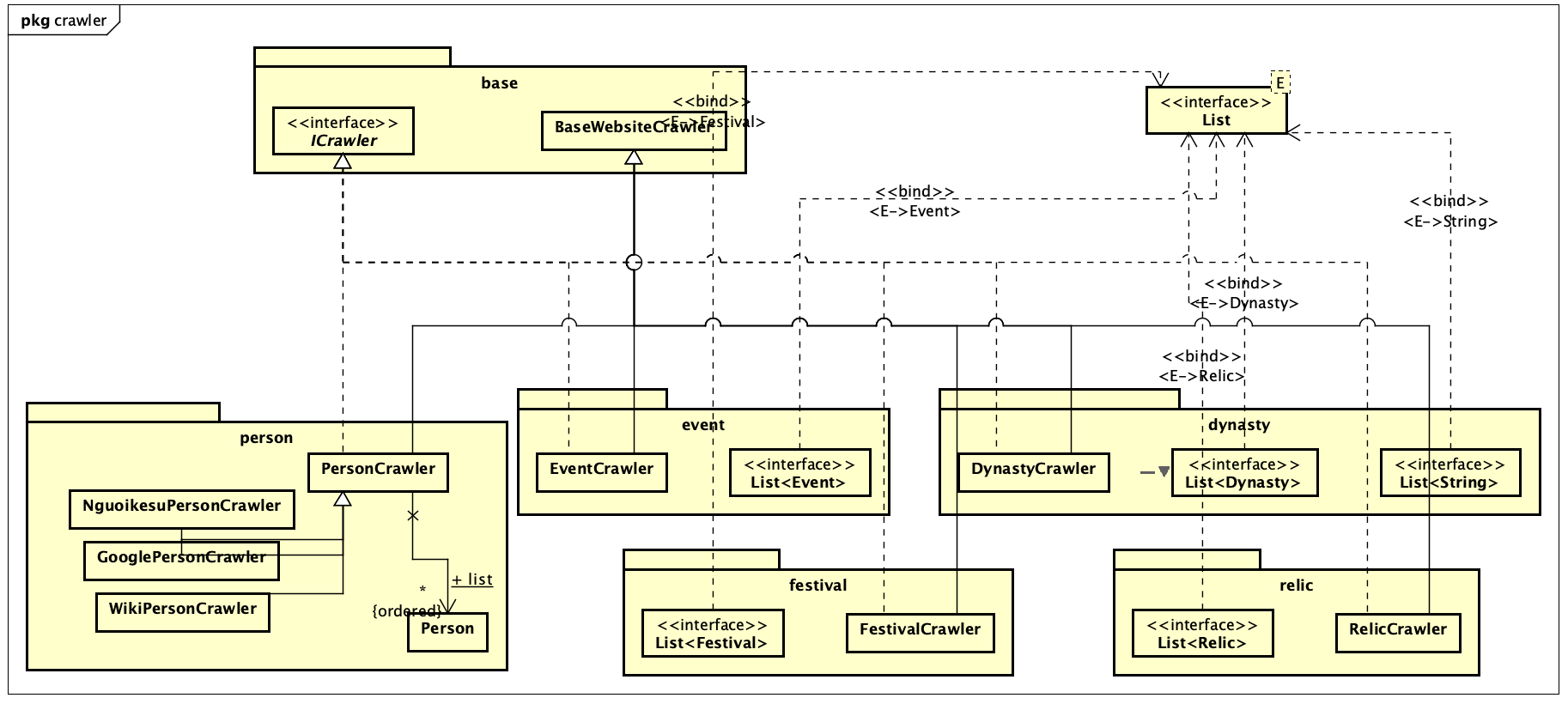
| Data | Count | Attributes | Description |
| --- | --- | --- | --- |
| Dynasty | 14 | id, name, existedTime, capital, kingName/ listofKing | The dynasties that existed throughout Vietnam history |
| Event | 71 | id, name, time, destination, description, image, relatedPerson | Wars, resignation, and other historical occurrences |
| Festival | 51 | id, name, time, destination, description | Special celebrations of Vietnam |
| Person | 1397 | id, name, givenName, father, dateOfBirth, dateOfDeath, description, dynasty | Public figures, historical heroes, and person related to history |
| Relic | 22 | id, title, content, address, relatedPerson | The historical monuments and places alike |

# UML Diagrams & Design Explanation

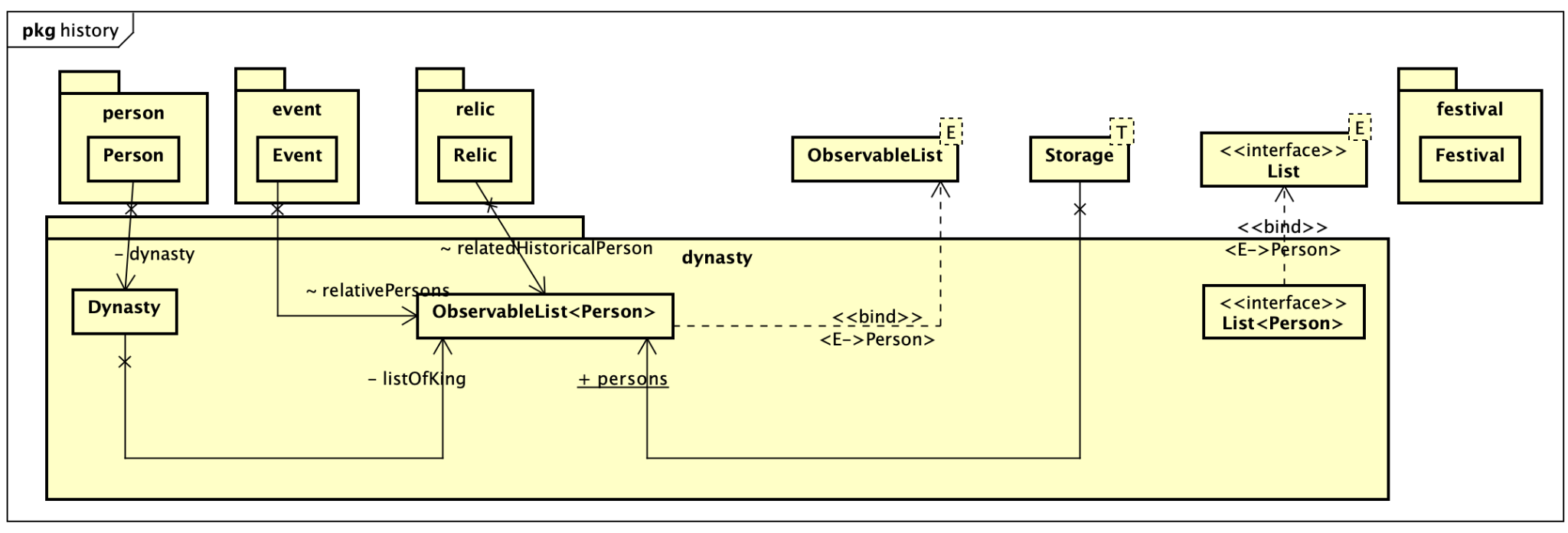
Some developers prefer to create UML diagrams before coding as part of the planning and design phase. However, we prefer to create UML diagrams after coding. This approach involves analyzing the existing codebase and generating UML diagrams to document the system's structure.

It can be useful for understanding complex or legacy codebases and identifying patterns. It also helps us to come up with improvements and suggest refactoring if needed.

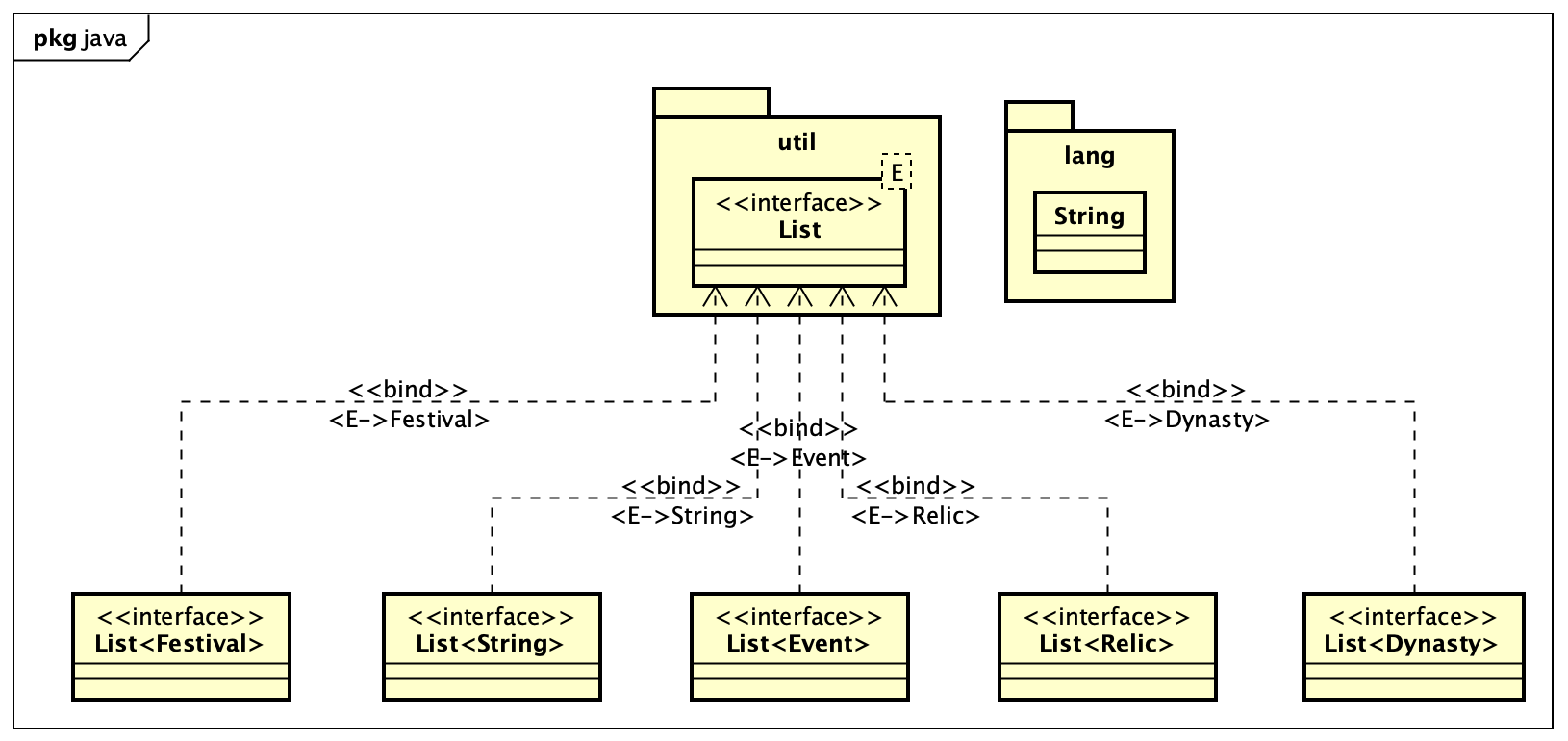
## Package Diagram



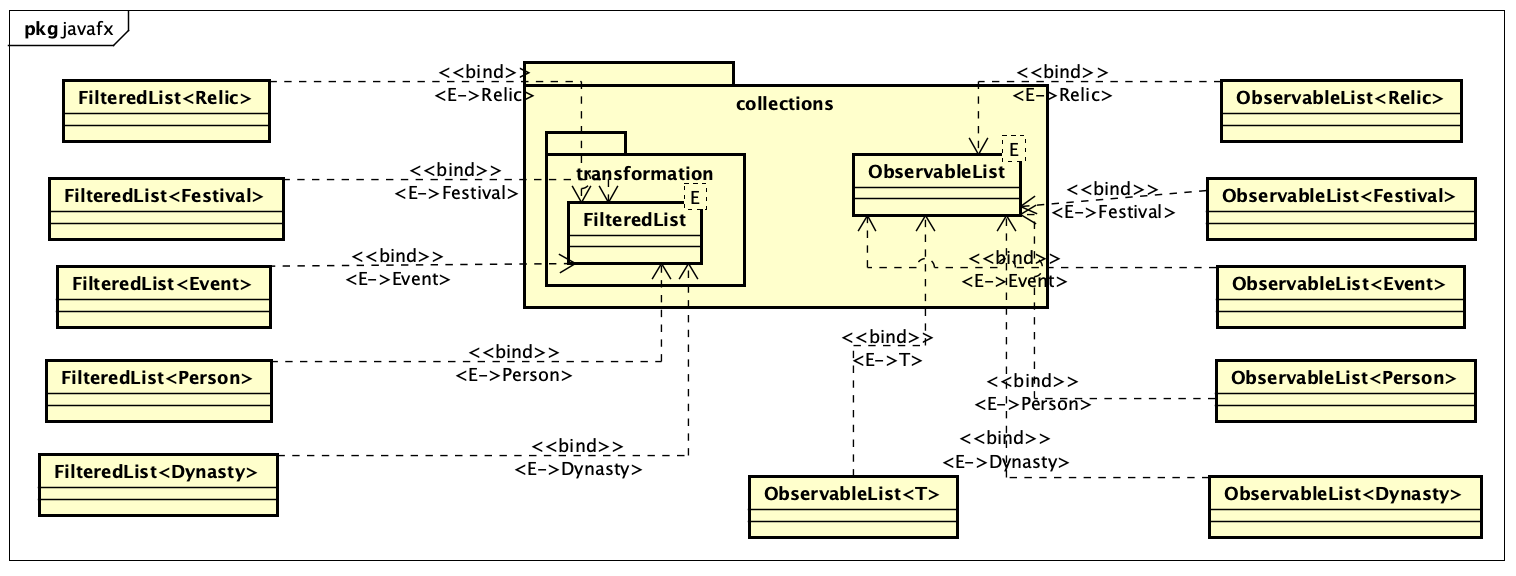
The package of crawlers. Here a certain crawler for an entity is grouped into its own package. The List interface in Java is also used.



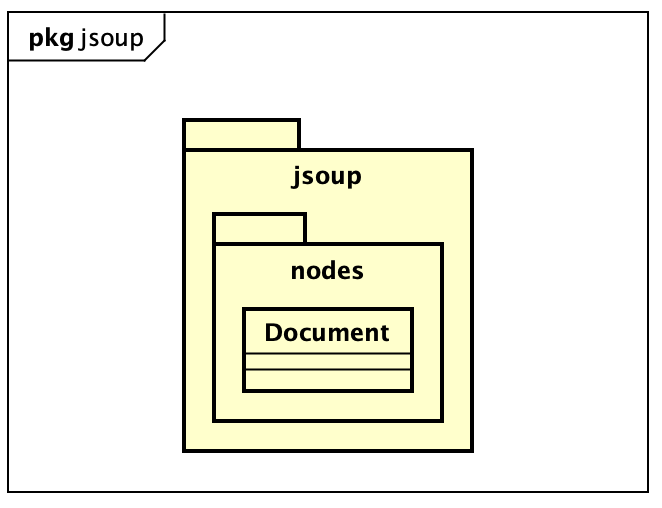
The package of classes for the entities. Each entity is in a package.



The package of the List Java built-in interface and class used.



The package of JavaFX collections. The data, now encapsulated in objects, are passed to the ObservableLists. The FilteredList provides the wrapper around the source list.

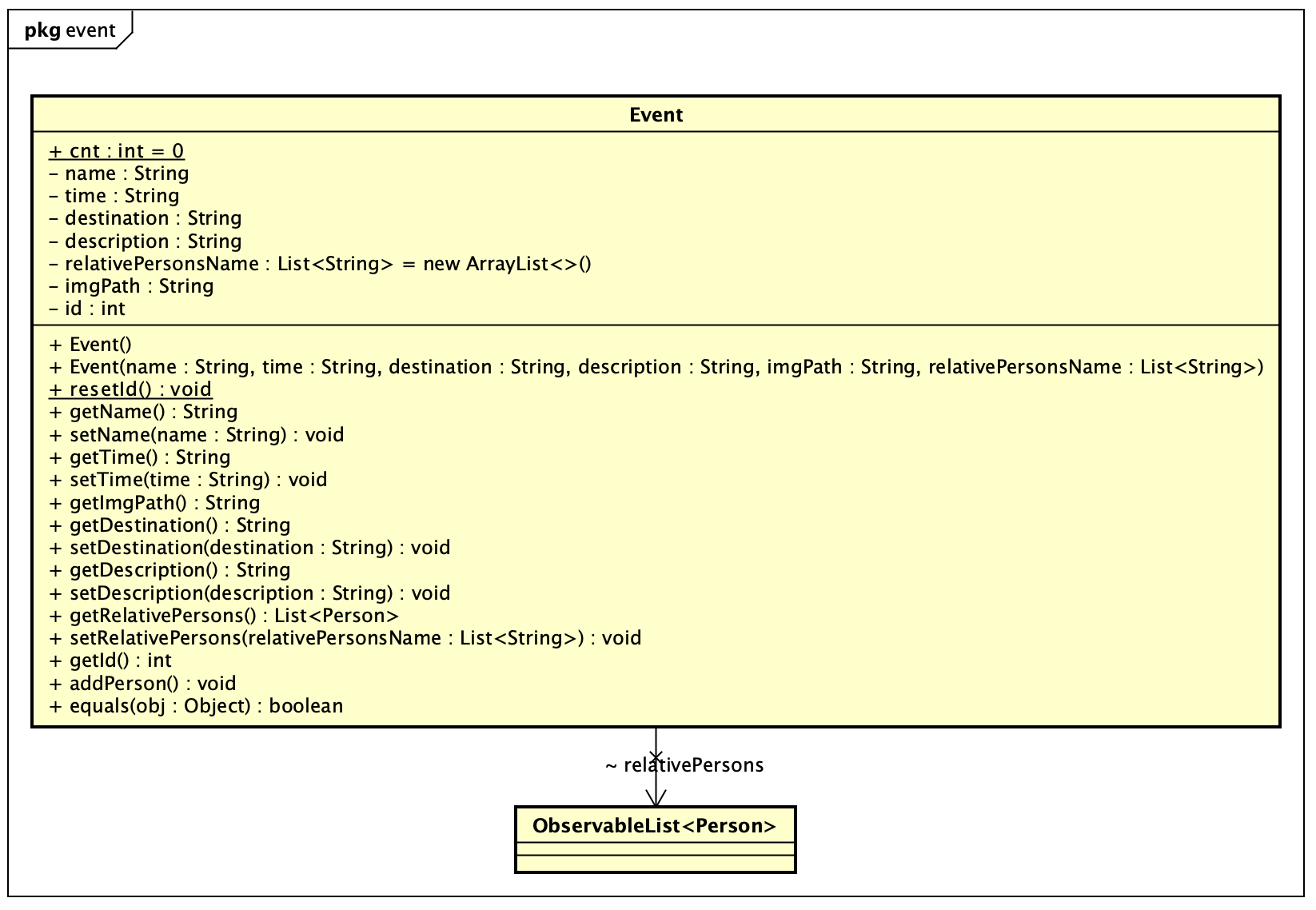


The package of Jsoup.

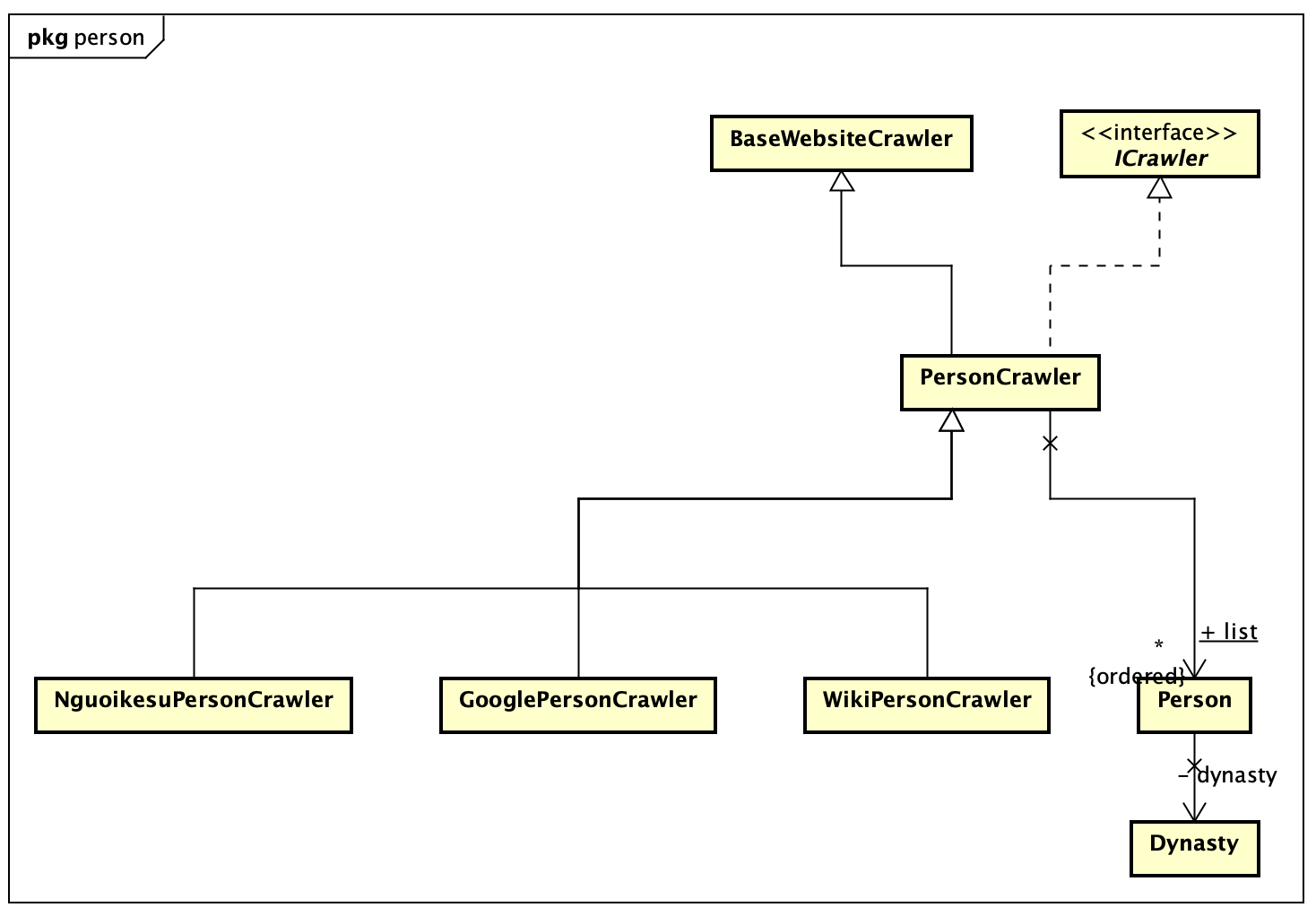
## Class Diagram

## 

The class diagram of storage objects.



The class diagram of entity Event. This is a class that has only one relation to another entity (Person).



The class diagram of the Person Crawler. This is an abstract class that inherits from the BaseCrawler, and there are source-specific classes that inherit from it to implement different scraping methods for each source.

# Applied OOP Techniques

Throughout the development of the project, we tried our best to apply the OOP techniques learned in class. Notably the 4 main principles of OOP are used:

## Encapsulation

This principle emphasizes the bundling of data and methods into a single unit called an object.

#### What we do

* Assign each entity to a class of its own (Dynasty, Event, Relic,...);
* Assign the crawler of each entity to a class of its own (DynastyCrawler, EventCrawler, RelicCrawler…);
* Grouping the UI controllers for each entity. For each entity, the controllers for the detailed information and the list interface are also broken down into separate ones. (DynastyDetailController, EventDetailController, RelicDetailController…) and (DynastyListController, EventListController,...)

#### The benefits

By encapsulating related data and behavior together, we can achieve data hiding and protect the internal state of an object. This enhances security, modularity, and code reusability.

## Inheritance

Inheritance allows us to create new classes based on existing classes, inheriting their attributes and behaviors.

#### What we do

* Implement an abstract class BaseCrawler that serves as the template for other crawlers. This class has attribute URL, crawl, write to json method that other crawlers based on. The crawlers for each entity inherits this base class.

#### The benefits

It promotes code reuse and enables the creation of hierarchical relationships between classes. With inheritance, we can build specialized classes that inherit common characteristics from a base class, reducing code duplication and improving maintainability.

## Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass.

#### What we do

* The crawlers override the method specified in the abstract base class;
* In each entity class, constructor overloading is used extensively to allow for different cases of data (i.e. when only some data field/ attribute is available)

#### The benefits

It enables us to write code that can work with objects of multiple types, providing flexibility and extensibility. Polymorphism allows for method overriding, where a subclass can provide its own implementation of a method defined in its superclass, further enhancing code modularity and flexibility.

## Abstraction

Abstraction involves simplifying complex systems by breaking them down into manageable and understandable components.

#### What we do

* Break down the essentials of the project:
  + An abstract class named "PersonCrawler" defines an abstract method named "crawl". This method is implemented by other types of crawler, depending on web structure and base URL of other classes.
  + “DetalBaseController” define method “initMainContent” that other classes(EventBaseController, RelicBaseController, FestivalBaseController, RelicBaseController, PersonBaseController) show detail content.
  + The app, which contains the backend working of:
    - The crawlers
    - The objects
    - The data crawled
    - The storage structure
  + The GUI, which contains the frontend working of:
    - The controllers
    - CSS
    - The JavaFX and FXML Scene Builder

#### The benefits

It allows us to focus on the essential features of an object or system while hiding unnecessary details. Abstraction helps in designing modular and maintainable code, as well as in creating reusable components.

When the user interacts with our app, they do not need to know the underlying structure of the program or understand how everything works.

Furthermore, other notable characteristics of our project are the application of some common design patterns, as well as the tech stack used.

# Design Pattern

## Factory Pattern

BaseWebsiteCrawler works as a factory for other types of constructor.

Benefits

The Factory Pattern encapsulates object creation, provides an abstraction layer, promotes flexibility and code reuse. It separates the object creation process from the client code and allows for changes in the creation process without affecting other classes.

## Strategy Pattern

In class Storage, to handle search functionality for different categories of data or different algorithms. In our project, we implement 5 methods of search(searchDynasty, searchFestival, searchPerson, searchRelic, searchEvent) based on string matching.

## Singleton Pattern

To ensure that only one instance of each crawler is running at a time, we implement the Singleton Pattern by making each crawler class a singleton.

The benefit of the Singleton Pattern to prevent conflicts between multiple instances of the same crawler running simultaneously, reduces the memory footprint of the application, as there is only one instance of each crawler class in memory at any given time.

# Tech Stack

1. Maven: The library classes added through Maven help to synchronize code (avoid conflicts between versions or IDEs).
2. Java Coding Convention: Classes, packages, variable names, and functions are named according to the standard.
3. Build Class and package diagrams using a consistent UML modeling language in Astah.
4. Jsoup: A specialized library for data collection through web pages, added through Maven.
5. Build a GUI interface with JavaFX Scene Builder.
6. Gson: A Google library for Java used for reading or writing data with .json files to objects.
7. Github: Managing source code through Github.

# User Guide

To get started, clone the source code from [this GitHub repository](https://github.com/Ashirog1/oop_history_prj). Make sure you have Java and associated softwares mentioned in the Tech Stack.

Navigate to the src folder of the code. From the App.java file, run the main() method.

Now you can interact with the app via GUI.

# GUI Highlight

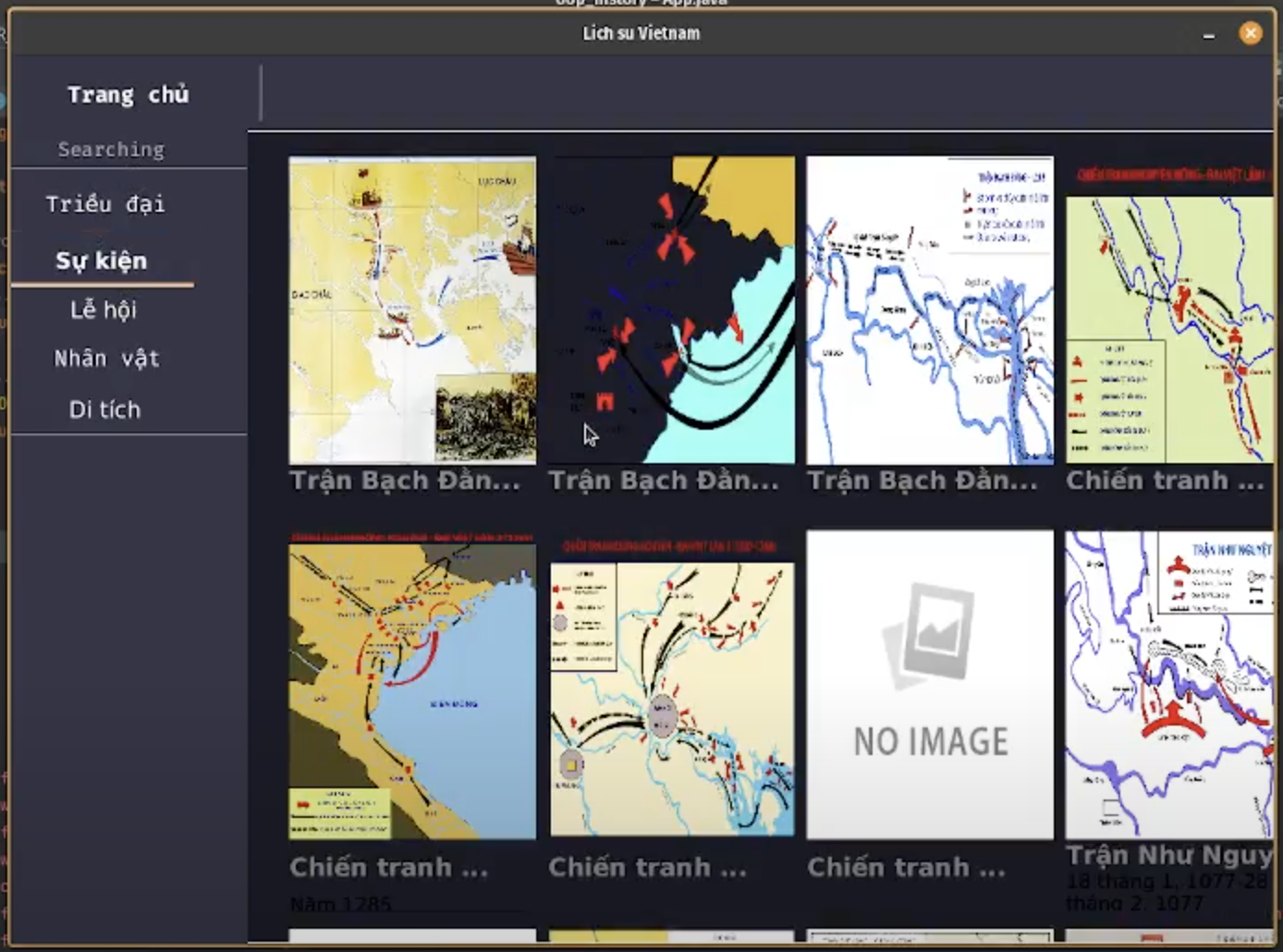
Here is a preview of the app interaction. It is fairly simple to navigate, though it can be better with further polishment.



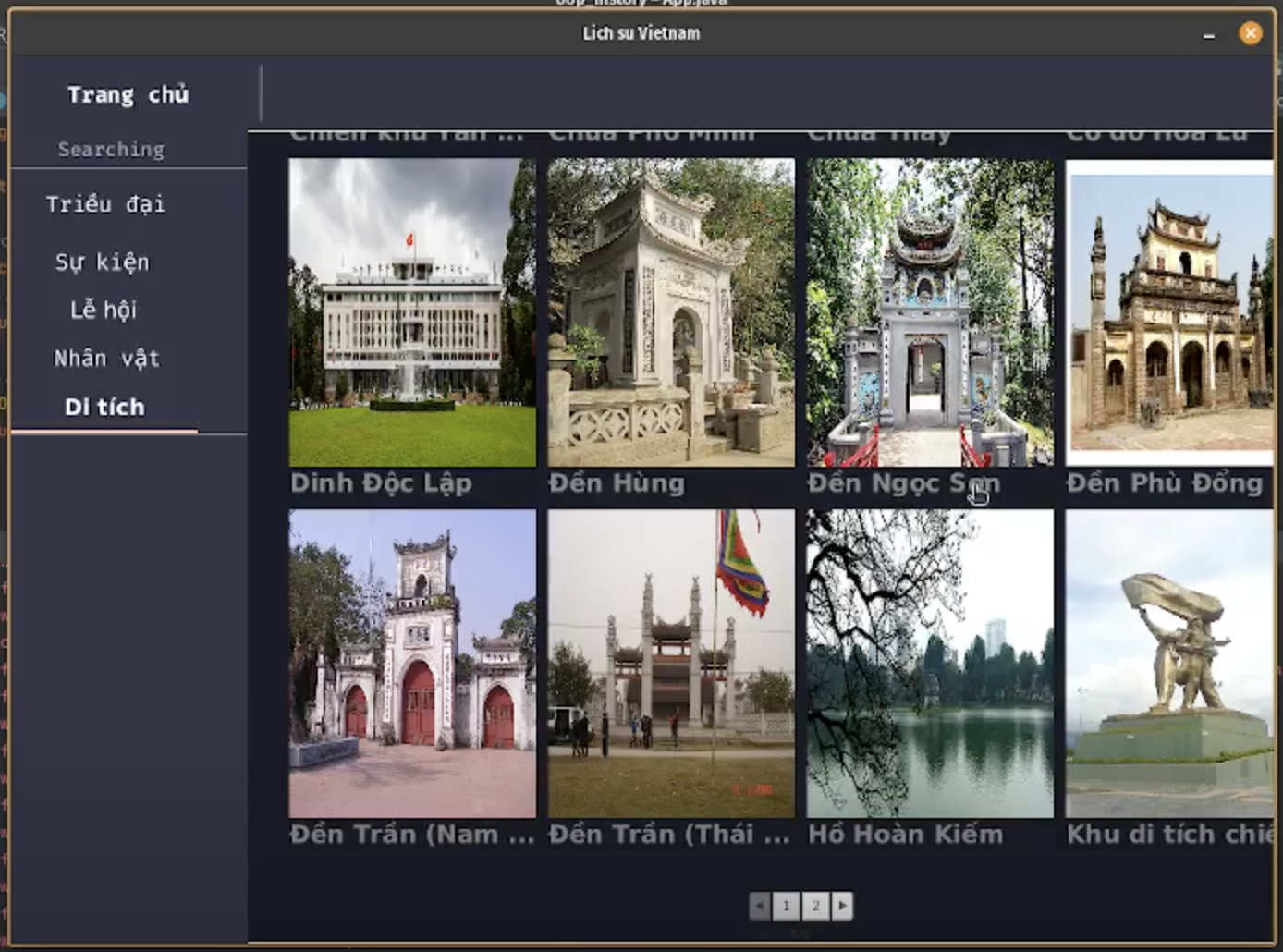
The onboarding screen.



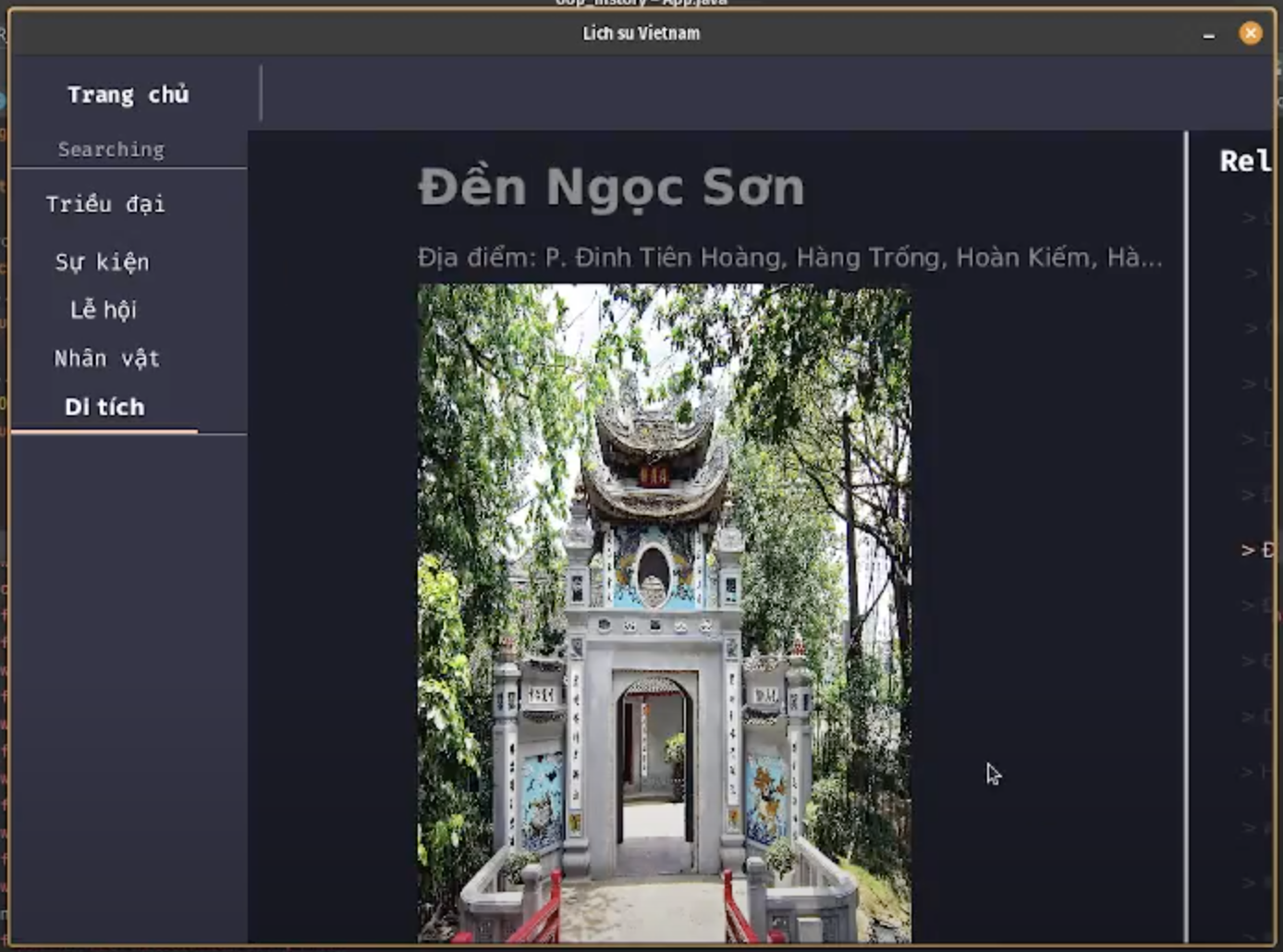
The dashboard. It selects the view of Dynasties by default. To navigate to another entity, look for the navigation bar from the left. There is also a search bar available on top of that.



The view of Events, which also have related images as part of each info card.



Excluding Dynasties, a similar view is available for all other entities.



A detailed view.

THE END