web scrap

1. Project overview

This project is a Java-based application for scraping job offers, storing them in a database, and performing machine learning on the collected data. The project integrates various technologies and concepts, including GUI development, web scraping, database interaction, and basic machine learning.

---

2. Technologies Used

- **GUI Framework:** JavaFX
- **Web Scraping:** Jsoup and Selenium
- **Database:** MySQL (accessed via JDBC)
- **Machine Learning:** Custom Java implementation (no external ML frameworks)

---

3. Proposed Project Structure

The project is divided into the following main packages to ensure modularity and maintainability:

1. `gui`: Contains the JavaFX application and controllers.
2. `scraper`: Handles web scraping for multiple websites.
3. `data`: Manages database interaction and data models (e.g., DAO classes, entity classes).
4. `service`: Contains the core business logic to link the scraper, database, and GUI.
5. `ml`: Implements machine learning functionalities.

---

4. Key Design Patterns

4.1 Scraper Interface

To ensure flexibility and scalability, a generic `Scraper` interface will be used. Each website scraper will implement this interface, providing specific logic for that website.

```
public interface Scraper {
List<Job> scrape(); // Returns a list of scraped job offers.
}
```

Implementation Example:

java

Copier le code

`public class Website1Scraper implements Scraper { @Override public List<Job> scrape() { // Logic for scraping Website1 return new ArrayList<>(); } }`

4.2 Database Access Object (DAO) Pattern

The `DAO` pattern abstracts database operations, making the code more modular and testable. The `JobDAO` class will handle all database interactions for the `Job` entity.

---

5. Workflow

Step 1: Scraping Job Offers

1. Each website scraper implements the `Scraper` interface.
2. A central `ScrapingService` class manages all the scrapers and aggregates results.

```
public class ScrapingService {
     private final List<Scraper> scrapers;
     public ScrapingService() {
               scrapers = List.of(new Website1Scraper(),new
Website2Scraper());
                   }
    public List<Job> scrapeAll() {
             List<Job> allJobs = new ArrayList<>();
             for (Scraper scraper : scrapers) {
                allJobs.addAll(scraper.scrape());
                }
                return allJobs;
```

```
            }
            }
```

Step 2: Storing Data in the Database

The `JobService` class:

1. Calls `ScrapingService` to fetch job offers.
2. Passes the data to `JobDAO` to store in the database.

Example:

```
public class JobService {
        private final JobDAO jobDAO;
        private final ScrapingService scrapingService;
        public JobService() {
            this.jobDAO = new JobDAO();
            this.scrapingService = new ScrapingService();
            }
        public void scrapeAndSaveJobs() {
             List<Job> jobs = scrapingService.scrapeAll();
                 for (Job job : jobs) {
                    jobDAO.insert(job);
                }
                }
}
```

Step 3: Displaying Data in the GUI

The GUI interacts with the `JobService` or directly with `JobDAO` to fetch stored data and display it.

---

6. Database Design

A simple table structure for the job offers:

Column	Type	Description
`id`	INT (Primary Key)	Unique identifier
`title`	VARCHAR(255)	Job title
`company`	VARCHAR(255)	Company name
`location`	VARCHAR(255)	Job location
`description`	TEXT	Job description

---

7. Key Classes

7.1 Entity Class: Job

Represents a job offer and is used across the project.

```
package data;
public class Job {
        private String title;
        private String company;
        private String location;
        private String description;
        public Job(String title, String company, String location, String
description){
        this.title = title;
        this.company = company;
        this.location = location;
        this.description = description;
        }
        // Getters and setters omitted for brevity
    }
```

7.2 DAO Class: JobDAO

Handles database interactions for the `Job` entity.

// code for handling db connection and sending the querry

---

8. Advantages of This Design

1. **Modularity:** Each component (GUI, scraping, database, ML) is independent.
2. **Scalability:** New scrapers can be added easily by implementing the `Scraper` interface.
3. **Maintainability:** Database logic is abstracted using the DAO pattern.
4. **Reusability:** Components like `JobDAO` and `ScrapingService` can be reused or extended.

---

9. Future Enhancements

1. **Error Handling:** Implement robust error handling for web scraping and database operations.
2. **Threading:** Use multithreading to scrape multiple websites concurrently.
3. **Testing:** Write unit tests for individual components (e.g., DAO methods, scraping logic).
4. **Configuration:** Use configuration files for managing database credentials, scraper settings, etc.
5. **Logging:** Add logging to monitor the scraping and database insertion process.

---

10. Summary Workflow

1. **Scraping Process:** GUI triggers scraping → `ScrapingService` coordinates scrapers → Returns a list of jobs.
2. **Database Process:** Jobs are saved to the database using `JobDAO`.
3. **Display Process:** GUI fetches and displays job data from the database.

This document provides a clear overview of the project's structure and workflow. Feel free to use it as a reference or to explain the project to your coworkers or teacher. Let me know if you'd like further clarifications!