Jaden Horst - 30432

Documentation for StackOverFlow Application

Overview

The StackOverFlow Application is a Spring Boot-based project designed to demonstrate CRUD operations and interactions between different entities such as `Answer`, `Asker`, `Expert`, and `Question`. It showcases the use of Spring Data JPA for database operations, Spring Web for RESTful services, and basic security configuration for API access. This documentation covers the primary components of the application, including models, repositories, services, and controllers.

Application Components

Models

`Answer`

- Represents an answer provided by an expert to a question.
- Attributes: `answerId`, `answerName`, and an association to `Expert`.

`Asker`

- Represents the individual who asks a question.
- Attributes: `cnp` (a unique identifier), `name`.

`Expert`

- Represents an expert who can answer questions.
- Attributes: `expertId`, `lastName`, `firstName`.

`Question`

- Represents a question asked by an asker.
- Attributes: `questionId`, `questionName`, and an association to `Expert`.

Repositories

Each model has a corresponding repository interface for CRUD operations, extending `CrudRepository` from Spring Data JPA:

- `AnswerRepository` for `Answer` entity.
- `AskerRepository` for `Asker` entity.
- `ExpertRepository` for `Expert` entity.
- `QuestionRepository` for `Question` entity.

Services

Services are provided for each entity to encapsulate the business logic:

- `AnswerService`
  - Methods: `retrieveAnswer()`, `insertAnswer(Answer answer)`, `deleteById(Long id)`.
- `AskerService`
  - Methods: `retrievePersons()`.
- `ExpertService`
  - Methods: `retrieveExperts()`, `retrieveExpertById(Long id)`, `addExpert(Expert expert)`, `deleteExpertById(Long id)`.
- `QuestionService`
  - Methods: `retrieveQuestion()`, `insertQuestion(Question question)`, `deleteById(Long id)`.

Controllers

Controllers expose the APIs to interact with the services:

`AnswerController`

- Base Path: `/answers`
- Endpoints:
  - GET `/getAll` - Retrieves all answers.
  - POST `/insertAnswer` - Inserts a new answer.
  - PUT `/updateAnswer` - Updates an existing answer.
  - DELETE `/deleteById` - Deletes an answer by ID.

`AskerController`

- Base Path: `/asker`
- Endpoints:
  - GET `/getAll` - Retrieves all askers.

`ExpertController`

- Base Path: `/experts`
- Endpoints:
  - GET `/getAll` - Retrieves all experts.
  - GET `/getById/{id}` and `/getById` - Retrieves an expert by ID.
  - POST `/createExpert` - Creates a new expert.
  - PUT `/updateExpert` - Updates an existing expert.
  - DELETE `/deleteById` - Deletes an expert by ID.

`QuestionController`

- Base Path: `/questions`
- Endpoints:
  - GET `/getAll` - Retrieves all questions.
  - POST `/createQuestion` - Creates a new question.
  - PUT `/updateQuestion` - Updates an existing question.
  - DELETE `/deleteById` - Deletes a question by ID.

Testing

Unit and integration tests are included to ensure the reliability of the application components. Test classes use Spring Boot Test, JUnit, Mockito, and AssertJ for comprehensive testing of services, controllers, and repositories.

Examples

- `AnswerControllerTest` and `ExpertControllerTest` demonstrate testing controllers using `MockMvc` to simulate HTTP requests and verify responses.
- `AnswerRepositoryTest` shows how to test repository interactions with an in-memory database using `DataJpaTest`.

Configuration

Application properties are defined in `application.properties`, including Spring's datasource URL, username, and password for database access.

Running the Application

To run the application, execute:

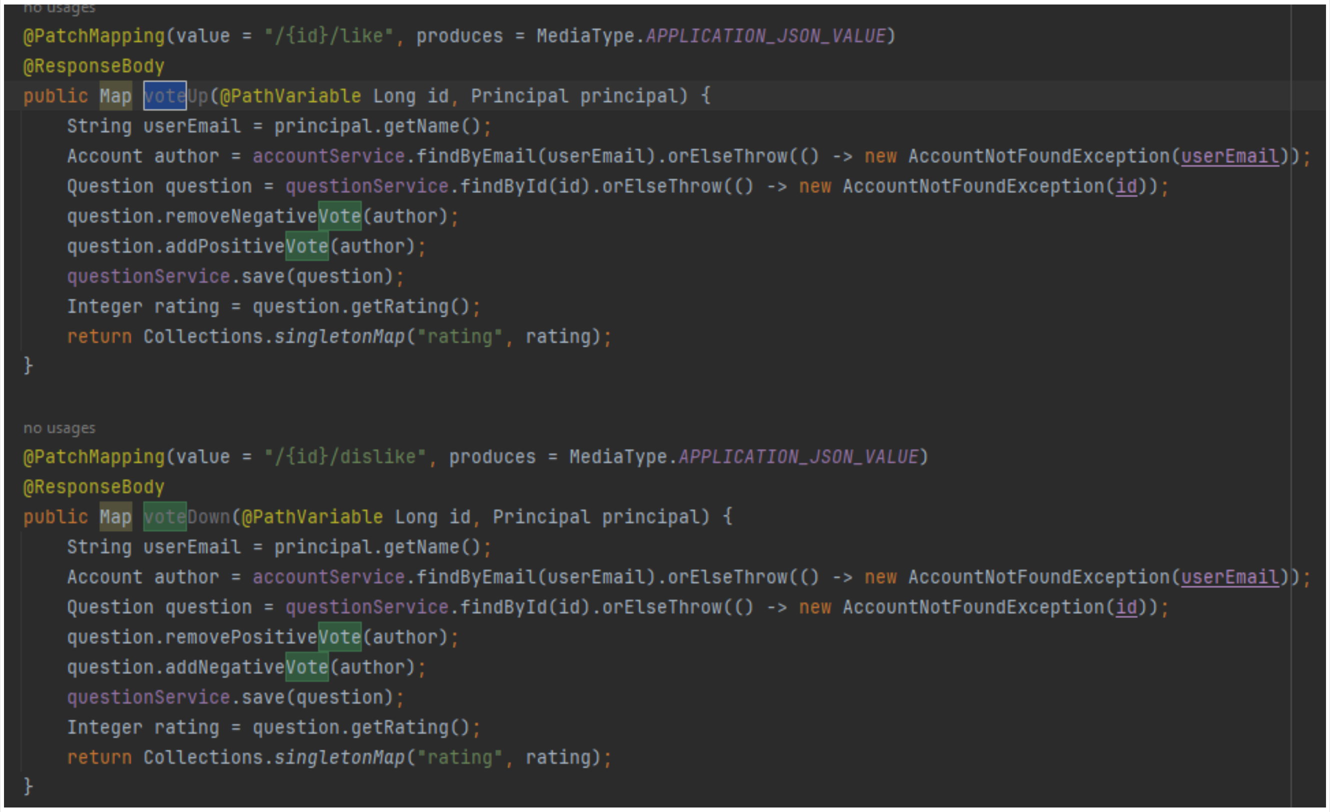
```
shellCopy code
spring-boot:run
```

This command starts the Spring Boot application, making the REST-ful APIs available for interaction.

Updates to Handle Votes and Images

QuestionController

The `QuestionController` class has been updated to include endpoints for voting on questions. Here is an updated snippet of the controller with new functionalities:



```
no usages
@PatchMapping(value = "/{id}/like", produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public Map voteUp(@PathVariable Long id, Principal principal) {
    String userEmail = principal.getName();
    Account author = accountService.findByEmail(userEmail).orElseThrow(() -> new AccountNotFoundException(userEmail));
    Question question = questionService.findById(id).orElseThrow(() -> new AccountNotFoundException(id));
    question.removeNegativeVote(author);
    question.addPositiveVote(author);
    questionService.save(question);
    Integer rating = question.getRating();
    return Collections.singletonMap("rating", rating);
}

no usages
@PatchMapping(value = "/{id}/dislike", produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public Map voteDown(@PathVariable Long id, Principal principal) {
    String userEmail = principal.getName();
    Account author = accountService.findByEmail(userEmail).orElseThrow(() -> new AccountNotFoundException(userEmail));
    Question question = questionService.findById(id).orElseThrow(() -> new AccountNotFoundException(id));
    question.removePositiveVote(author);
    question.addNegativeVote(author);
    questionService.save(question);
    Integer rating = question.getRating();
    return Collections.singletonMap("rating", rating);
}
```

Conclusion

The StackOverFlow Application exemplifies a simple yet effective structure for a Spring Boot project, facilitating learning and exploration of Spring's core functionalities, including web services, data access, and basic security configurations.