Backend Documentation for Movie Review Application

1. Overview

This repository contains the backend code for a Movie Review application developed using **Spring Boot** and **MongoDB**. The goal is to provide APIs for managing movies, fetching details, and creating/retrieving reviews.

2. Project Structure

```
/src
 ├── /main
 │    ├── /java
 │    │    ├── com
 │    │    │    ├── ujjwal
 │    │    │    │    ├── Movies
 │    │    │    │    │    ├── Movie.java           # Movie entity class
 │    │    │    │    │    ├── MovieController.java  # Controller for movie-related
endpoints
 │    │    │    │    │    ├── MovieService.java     # Service layer for business
logic related to movies
 │    │    │    │    │    ├── MovieRepository.java  # Repository for MongoDB
operations related to movies
 │    │    │    │    │    ├── Review.java           # Review entity class
 │    │    │    │    │    ├── ReviewController.java # Controller for review-
related endpoints
 │    │    │    │    │    ├── ReviewService.java    # Service layer for business
logic related to reviews
 │    │    │    │    │    └── ReviewRepository.java # Repository for MongoDB
operations related to reviews
 │    │    │    │    │
 │    │    │    ├── config                         # Configuration classes
 │    │    │    │    └── WebConfig.java            # CORS configuration
 │    │    │    └── Application.java               # Main Spring Boot application
entry point
 │    │    └── resources
 │    │         └── application.properties      # Spring Boot configuration
 └── /test
      └── com
           └── ujjwal
                └── Movies
                     └── MovieControllerTest.java  # Unit tests for
MovieController
                     └── ReviewControllerTest.java # Unit tests for
ReviewController
```

---

3. Key Components

1. `Movie` Entity Class (`Movie.java`)

- Represents a Movie document stored in MongoDB.
- Fields include:

    - `id` (ObjectId): Auto-generated MongoDB ID.
    - `imdbId`: Unique identifier for the movie.
    - `title`: Title of the movie.
    - `releaseDate`: Release date of the movie.
    - `trailerLink`: URL for the movie trailer.
    - `poster`: URL for the movie poster.
    - `backdrops`: List of backdrop image URLs.
    - `genres`: List of movie genres.
    - `reviews`: List of reviews associated with the movie.

```
@Document(collection="Movie_data")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Movie {
    @Id
    private ObjectId id;
    private String imdbId;
    private String title;
    private String releaseDate;
    private String trailerLink;
    private String poster;
    private List<String> backdrops;
    private List<String> genres;

    @DocumentReference
    private List<Review> reviews;
}
```

---

2. `MovieRepository` (`MovieRepository.java`)

- Interface for MongoDB CRUD operations on the `Movie` collection.
- Custom method to find a movie by its `imdbId`.

```
public interface MovieRepository extends MongoRepository<Movie, ObjectId> {
    Optional<Movie> findMovieByImdbId(String imdbId);
}
```

3. `MovieService` (`MovieService.java`)

- Handles the business logic related to movies.
- `findAllMovies()` - Fetches all movies from the database.
- `findMovieByImdbId(String imdbId)` - Fetches a movie by its `imdbId`.

```
@Service
public class MovieService {
    @Autowired
    private MovieRepository movierepo;

    public List<Movie> findAllMovies() {
```

```
        return movierepo.findAll();
    }

    public Optional<Movie> findMovieByImdbId(String imdbId) {
        return movierepo.findMovieByImdbId(imdbId);
    }
}
```

4. `MovieController` (`MovieController.java`)

- REST controller to handle incoming requests related to movies.
- Provides endpoints:
    - `GET /api/v1/movies`: Fetch all movies.
    - `GET /api/v1/movies/{imdbId}`: Fetch a movie by its `imdbId`.

```
@RestController
@RequestMapping("/api/v1/movies")
public class MovieController {
    @Autowired
    private MovieService movieservice;

    @GetMapping
    public ResponseEntity<List<Movie>> getMovies() {
        return new ResponseEntity<>(movieservice.findAllMovies(),
HttpStatus.OK);
    }

    @GetMapping("/{imdbId}")
    public ResponseEntity<Optional<Movie>> getSingleMovie(@PathVariable
String imdbId) {
        return new ResponseEntity<>(movieservice.findMovieByImdbId(imdbId),
HttpStatus.OK);
    }
}
```

---

5. `Review` Entity Class (`Review.java`)

- Represents a Review document stored in MongoDB.
- Fields:
    - `id` (ObjectId): Auto-generated MongoDB ID.
    - `body`: The review content.
    - `created`: Timestamp when the review was created.
    - `updated`: Timestamp when the review was last updated.

```
@Document(collection = "Reviews")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Review {
    @Id
    private ObjectId id;
```

```
    private String body;
    private LocalDateTime created;
    private LocalDateTime updated;

    public Review(String body, LocalDateTime created, LocalDateTime updated)
{
        this.body = body;
        this.created = created;
        this.updated = updated;
    }
}
```

---

6. `ReviewRepository` (`ReviewRepository.java`)

- Interface for MongoDB CRUD operations on the `Review` collection.

```
public interface ReviewRepository extends MongoRepository<Review, ObjectId> {
}
```

---

7. `ReviewService` (`ReviewService.java`)

- Handles business logic related to reviews.
- `createReview(String reviewBody, String imdbId)` - Creates a new review and associates it with the movie.

```
@Service
public class ReviewService {
    @Autowired
    private ReviewRepository repository;

    @Autowired
    private MongoTemplate mongoTemplate;

    public Review createReview(String reviewBody, String imdbId) {
        Review review = repository.insert(new Review(reviewBody,
LocalDateTime.now(), LocalDateTime.now()));

        mongoTemplate.update(Movie.class)
                .matching(Criteria.where("imdbId").is(imdbId))
                .apply(new Update().push("reviews").value(review.getId()))
                .first();

        return review;
    }
}
```

---

8. `ReviewController` (`ReviewController.java`)

- REST controller to handle incoming requests related to reviews.
- Provides the endpoint:

- `POST /api/v1/reviews`: Create a new review.

```
@RestController
@RequestMapping("/api/v1/reviews")
public class ReviewController {
    @Autowired
    private ReviewService service;

    @PostMapping
    public ResponseEntity<Review> createReview(@RequestBody Map<String,
String> payload) {
        String reviewBody = payload.get("reviewBody");
        String imdbId = payload.get("imdbId");

        Review review = service.createReview(reviewBody, imdbId);

        return new ResponseEntity<>(review, HttpStatus.OK);
    }
}
```

---

4. Configuration (`WebConfig.java`)

- CORS configuration to allow cross-origin requests from specified origins.
- Enables credentials and specific allowed HTTP methods for frontend access.

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("http://localhost:3000",
"https://your-ngrok-url.com")
                        .allowedMethods("GET", "POST", "PUT", "DELETE",
"OPTIONS")
                        .allowedHeaders("*")
                        .allowCredentials(true);
            }
        };
    }
}
```

---

5. MongoDB Configuration

- The application uses MongoDB to store `Movie` and `Review` data.
- Make sure MongoDB is running, and update the connection settings in `application.properties`:

```
spring.data.mongodb.uri=mongodb://localhost:27017/moviesDB
```

```
spring.mongodb.database=moviesDB
```

---

6. Running the Application

1. **Build the project** using Maven or Gradle.
2. **Run** the `MoviesApplication.java` file as a Spring Boot application.
3. Access the API via the configured endpoints to interact with movies and reviews.

---

7. Endpoints Overview

- **Movies API**:
  - `GET /api/v1/movies` - Fetch all movies.
  - `GET /api/v1/movies/{imdbId}` - Fetch a movie by `imdbId`.
- **Reviews API**:
  - `POST /api/v1/reviews` - Create a new review.
