Hier is een stap-voor-stap handleiding voor het ontwikkelen van een Spring Reactive Blog Platform dat voldoet aan de beschreven vereisten:

Stap 1: Maak een nieuw Spring Boot-project aan

1. **Gebruik Spring Initializr** (https://start.spring.io/ (https://start.spring.io/)):
    - Kies **Maven** als project.
    - Kies **Java** als taal.
    - Voeg de volgende dependencies toe:
        - Spring Boot DevTools
        - Spring Reactive Web (tegenhanger van Spring Web)
        - Spring Data Reactive MongoDB (tegenhanger van Spring Data JPA voor MongoDB)
        - Spring Boot Actuator
2. **Download het project** en open het in IntelliJ IDEA.

Stap 2: Configuratie van MongoDB

1. **Zorg ervoor dat MongoDB draait**:
    - Gebruik Docker om MongoDB op te starten:

      ```
      docker run --rm -d --name my-mongo -p 27017:27017 mongo mongod --replSet myReplicaSet
      docker exec my-mongo mongo mongosh --eval "rs.initiate();"
      ```

    - Voeg de MongoDB-configuratie toe aan `application.properties`:

      ```
      spring.data.mongodb.uri=mongodb://localhost:27017/blogdb
      logging.level.org.springframework.data.mongodb.core.ReactiveMongoTemplate=DEBUG
      ```

2. **Verbind MongoDB met IntelliJ**:
    - Ga naar het Database Tool Window en voeg MongoDB toe als datasource.

Stap 3: Het BlogPost-model maken

1. Maak een nieuwe klasse `BlogPost` die als document in MongoDB wordt opgeslagen:

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class BlogPost {
    @Id
    private String id;
    private String title;
    private String content;

    // Constructors, getters, setters
}
```

2. Voeg een repository toe voor de `BlogPost`:

```
import org.springframework.data.mongodb.repository.ReactiveMongoRepository;

public interface BlogPostRepository extends ReactiveMongoRepository<BlogPost, String> {
}
```

Stap 4: Voeg de Service-laag toe

1. Maak een service voor CRUD-bewerkingen op blogposts:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Mono;
import reactor.core.publisher.Flux;

@Service
public class BlogPostService {

    @Autowired
    private BlogPostRepository repository;

    public Flux<BlogPost> getAllBlogPosts() {
        return repository.findAll();
    }

    public Mono<BlogPost> getBlogPostById(String id) {
        return repository.findById(id);
    }

    public Mono<BlogPost> createBlogPost(BlogPost blogPost) {
        return repository.save(blogPost);
    }

    public Mono<Void> deleteBlogPost(String id) {
        return repository.deleteById(id);
    }

    public Mono<BlogPost> updateBlogPost(String id, BlogPost blogPost) {
        return repository.findById(id)
            .flatMap(existingBlogPost -> {
                existingBlogPost.setTitle(blogPost.getTitle());
                existingBlogPost.setContent(blogPost.getContent());
                return repository.save(existingBlogPost);
            });
    }
}
```

Stap 5: Voeg de REST API-controller toe

1. Maak een REST API-controller voor blogposts:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import reactor.core.publisher.Mono;
import reactor.core.publisher.Flux;

@RestController
@RequestMapping("/api/posts")
public class BlogPostController {

    @Autowired
    private BlogPostService blogPostService;

    @GetMapping
    public Flux<BlogPost> getAllPosts() {
        return blogPostService.getAllBlogPosts();
    }

    @GetMapping("/{id}")
    public Mono<BlogPost> getPostById(@PathVariable String id) {
        return blogPostService.getBlogPostById(id);
    }

    @PostMapping
    public Mono<BlogPost> createPost(@RequestBody BlogPost blogPost) {
        return blogPostService.createBlogPost(blogPost);
    }

    @PutMapping("/{id}")
    public Mono<BlogPost> updatePost(@PathVariable String id, @RequestBody BlogPost blogPost) {
        return blogPostService.updateBlogPost(id, blogPost);
    }

    @DeleteMapping("/{id}")
    public Mono<Void> deletePost(@PathVariable String id) {
        return blogPostService.deleteBlogPost(id);
    }
}
```

Stap 6: Voeg streaming functionaliteit toe

1. Voeg een endpoint toe om blogposts te streamen met een vertraging van 1 seconde:

```
@GetMapping("/stream/posts")
public Flux<BlogPost> streamBlogPosts() {
    return blogPostService.getAllBlogPosts()
            .delayElements(Duration.ofSeconds(1));
}
```

2. Voeg een event stream toe om blogpost-wijzigingen te streamen:

- Dit kan door MongoDB Change Streams te gebruiken:

```
@GetMapping("/stream/changes")
public Flux<String> streamChanges() {
    return blogPostService.getChangeStream();
}
```

Stap 7: Testen van de API

- Gebruik Postman om de verschillende endpoints te testen.
- Test de streaming API door verschillende waarden in de `Accept` header te gebruiken, zoals `application/json`, `text/event-stream`, en `application/x-ndjson`.

Stap 8: Web Applicatie

1. Zet de statische HTML en JS-bestanden onder `/src/main/resources/static`.
2. Maak een eenvoudige frontend om de blogposts weer te geven en interactie met de API te hebben.

Stap 9: Beveiliging met Spring Security

1. Voeg Spring Security toe om alleen admin-gebruikers blogposts te laten toevoegen of aanpassen:

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/api/posts/**").hasRole("ADMIN")
            .anyRequest().permitAll()
            .and()
            .httpBasic();
    }
}
```

2. Voeg rollen en gebruikers toe in de configuratie en zorg ervoor dat alleen admins de CRUD-bewerkingen kunnen uitvoeren.

Stap 10: Integration Tests

1. Schrijf integration tests voor de streaming endpoints en de beveiliging.

Conclusie

Met deze stappen kun je een Spring Reactive Blog Platform ontwikkelen met MongoDB en de nodige streamingfunctionaliteit. Zorg ervoor dat je de applicatie grondig test en de beveiliging correct implementeert.