Spring (REST)

Projectstructuur

- @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
- @ComponentScan: enable @Component scan on the package where the application is located
- @Configuration: allow to regiser extra beans in the context
- **@SpringBootApplication**: gelijk aan de hierbovenstaande gecombineerd

Dependency Injection

- **@Autowired**

DAO-klasse

- Annoteren met **@Service**
- HashMap aanmaken die het id op een object mapt
- Id's toewijzen aan objecten: static counter maken en initialiseren op een beginwaarde; telkens een object aangemaakt wordt --> counter++
- Alle objecten teruggeven --> lijst (of andere collectie) van map.values()

Controller-klasse

- Annoteren met **@RestController**
- DAO-klasse hier als instantievariabele zetten en met **@Autowired** annoteren

GET-request

- **@GetMapping("/naam")**
- **@GetMapping("/naam/{parameter})**
- Parameter oproepen in functie: public Returntype functieNaam(**@PathVariable("parameter")** type parameter)
- Indien er een parameter is die iets kan teruggeven dat errors geeft:

```
// In DAO
public Optional<Objet> getObject(type id) {
    return Optional.ofNullable(objects.get(id));
}
// In Controller
@GetMapping("/objects/{id}")
public BlogPost getObject(@PathVariable("id") type id) {
    return objectDao.getObject(id).orElseThrow(() -> new
ObjectNotFoundException(id));
}
```

POST-request

- **@PostMapping("/naam")**
- bijhorende functie heeft als returntype ResponseEntity met type Void
- **@RequestBody(Object object)** als parameter van de bijhorende functie
- URL teruggeven:
  URI location = ServletUriComponentsBuilder.fromCurrentRequestUri().path("/{parameter}")
  .buildAndExpand(functie_voor_parameter).toUri()
- return ResponseEntity.created(location).build();

DELETE-request

- **@DeleteMapping("/naam/{parameter}")**
- **@PathVariabele("parameter")**

PUT-request

- **@PutMapping("/naam/{parameter}")**
- **@PathVariabele("parameter")**

HTTP-code teruggeven

- **@ResponseStatus(HttpStatus.NAAM)**
- Indien er een andere code moet teruggegeven worden, dan de geannoteerde:

```
throw new ResponseStatusException(HttpStatus.NAAM, "_reden_");
```

ExceptionHandler

- **@ResponseStatus(HttpStatus.NOT_FOUND)** of een andere HTTP-code
- **@ExceptionHandler(ExceptieKlasse.class)**

Spring (JPA)

Dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

- Om gegevens in XML te kunnen tonen

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

Actuator

- De 4 golden signals (volgens Google) kunnen hiermee gemonitord worden

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

- Endpoints kiezen (via application.properties):
  management.endpoints.web.exposure.include=health,info,metrics

Testen API

- Web(Test)Client: **@SpringBootTest**-annotatie

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <scope>test</scope>
</dependency>
```

- Volledige annotatie:

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
```

- Elke methode met @Test annoteren
- DependencyInjection van de WebTestClient om te communiceren met de REST API

```
@LocalServerPort
private int port;

@Autowired
private WebTestClient webClient;
```

- Voorbeeld test om gegevens op te halen

```
@Test
public void haalBlogPostsOp() {
    webClient.get()
```

```
                .uri("/posts")
                .header(ACCEPT, APPLICATION_JSON_VALUE)
                .exchange()
                .expectStatus().isOk()
                .expectHeader().contentType(APPLICATION_JSON)
                .expectBodyList(BlogPost.class).hasSize(1)
                .contains(BlogPostDaoMemory.helloWorldPost);

    }
```

Datalaag

- In het labo hebben we H2 gebruikt als database om de gegevens in op te slaan

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

- Volgende toevoegen aan properties om database te kunnen bekijken:

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=
```

DAO-klasse

- Interface die de functies definieert
- Specifieke klasses die de interface implementeren
    - Annoteren met **@Service**
    - Annoteren met **@Profile(profiel)**
        - Afhankelijk indien deze DAO voor testen gebruikt wordt of niet
    - Bevat een repository (private final) -> dependency injection

Repository

- extends JpaRepository<type object, type id>

Objectklasse

- Annoteren met **@Entity**
- Annoteren met **@Table(name="naam")** (link leggen met tabel in DB)
- We moeten niet langer zelf een ID genereren; gewoon ID annoteren met:
    - **@Id**

- **@GeneratedValue**

Spring (Beveiliging)

Dependecy

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Beveiligingsklasse

- **@Configuration**
- **@EnableMethodSecurity(securedEnabled = true)**
- Credentials in properties (of zelf andere waardes te kiezen):

```
users.admin.username=admin
users.admin.password=admin
users.admin.roles=ADMIN
users.admin.encoded_password=
{bcrypt}$2a$10$/3zZXbQGPDvyZlTx2SSi.uBFLBqj/Dwc0goY9CvVjRKuH8AQ2NM/m
```

- Authenticatie toevoegen

```
@Value("${users.admin.password}")
private String adminPassword;

@Value("${users.admin.username}")
private String adminUsername;

@Value("${users.admin.roles}")
private String adminRoles;

@Value("${users.admin.encoded_password}")
private String adminEncodedPassword;
```

- Environment toevoegen:
  @Autowired
  Environment naam

- H2-database toevoegen om gegevens om te slaan

```
@Bean
public DataSource datasource() {
    return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.H2)
            .addScript(JdbcDaoImpl.DEFAULT_USER_SCHEMA_DDL_LOCATION)
            .build();
};
```

- Configuratie voor jdbc-authenticatie

```
@Bean
public UserDetailsManager users(DataSource datasource) {
    PasswordEncoder encoder =
PasswordEncoderFactories.createDelegatingPasswordEncoder();
    LoggerFactory.getLogger(SecurityConfig.class).info("Encoded password: " +
encoder.encode(adminPassword));
    UserDetails admin =
User.withUsername(adminUsername).password(encoder.encode(adminPassword)).roles(adm
inRoles).build();
    // Better use an externally hashed password to avoid clear text passwords in
source or memory
    // https://docs.spring.io/spring-
security/reference/features/authentication/password-storage.html#authentication-
password-storage-boot-cli
    UserDetails admin2 =
User.withUsername("admin2").password(adminEncodedPassword).roles(adminRoles).build
();
    JdbcUserDetailsManager users = new JdbcUserDetailsManager(datasource);
    users.createUser(admin);
    users.createUser(admin2);
    return users;
}
```

- SecurityFilterChain toevoegen

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.authorizeHttpRequests(requests -> requests
            .requestMatchers(new
AntPathRequestMatcher("/admin.html")).hasRole("ADMIN")
            .anyRequest().permitAll())
        .httpBasic(Customizer.withDefaults())
        .csrf(csrf -> csrf


.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
```

```
                .ignoringRequestMatchers(new AntPathRequestMatcher("/h2-
console/**"))
                .csrfTokenRequestHandler(new CsrfTokenRequestAttributeHandler()))
// Disable BREACH
        .headers(headers -> headers
                .frameOptions(frameOptionsConfig ->
frameOptionsConfig.sameOrigin()));
    if (environment.getActiveProfiles().length > 0 &&
environment.getActiveProfiles()[0].trim().equalsIgnoreCase("test")) {
        http.csrf(csrf -> csrf.disable());
    }
    return http.build();
}
```

- Om bepaalde methodes af te schermen met beveiliging
  - In de Controller-klasse
  - **@PreAuthorize("functie")**
  - Voorbeeld: **@PreAuthorize("hasRole('ROL')")**

Spring Reactive

Datalaag

Objectklasse

- Annoteren met **@Document("tabel")** met tabel de naam van de databasetabel
- Instantievariabele die het id voorstelt annoteren met **@Id**

Repository

- public interface ObjectRepository extends ReactiveMongoRepository<Object, type id>
- Alternatieven voorhanden indien er geen MongoDB gebruikt wordt
- Als de queries gelogd moeten worden in de console, volgende lijn toevoegen aan properties:

```
logging.level.org.springframework.data.mongodb.core.ReactiveMongoTemplate=DEBUG
```

REST API

DAO-klasse

- Delete-operaties hebben als returntype `Mono<Void>`
- Functies die één enkel object teruggeven hebben als returntype `Mono<Object>`
- Functies die meerdere objecten teruggeven hebben als returntype `Flux<Object>`
- Post- en Put-operaties hebben als returntype `Mono<Object>`

Stream teruggeven met x-aantal seconden vertraging tussen elk object

```
// in DAO-klasse

@Override
    public Flux<BlogPost> getChangeStreamPosts() {
        return reactiveMongoTemplate // dependency injection in constructor
                .changeStream(BlogPost.class)
                .watchCollection("posts")

.filter(Criteria.where("operationtype").in("insert","replace","update"))
                .listen()
                .mapNotNull(ChangeStreamEvent::getBody);
    }

// in Controller klasse

@GetMapping("/stream/objects-delay")
public Flux<Object> getPostsStreamV1() {
    this.objectsReadCounter.increment();
    return objectDAO.getAllPosts().delayElements(Duration.ofSeconds(1)).log();
}

@GetMapping(value = "/stream/objects-text", produces =
MediaType.TEXT_EVENT_STREAM_VALUE)
public Flux<Object> getPostsStreamV2() {
    this.objectsReadCounter.increment();
    return objectDAO.getAllPosts().delayElements(Duration.ofSeconds(1)).log();
}

@GetMapping(value = "/stream/objects-json", produces =
MediaType.APPLICATION_NDJSON_VALUE)
public Flux<Object> getPostsStreamV3() {
    this.objectsReadCounter.increment();
    return objectDAO.getAllPosts().delayElements(Duration.ofSeconds(1)).log();
}

/* ChangeStream of all objects. */
@GetMapping(value = "/stream/objects", produces =
MediaType.TEXT_EVENT_STREAM_VALUE)
public Flux<Object> getChangeStreamPosts() {
    this.objectsReadCounter.increment();
    return objectDAO.getChangeStreamPosts().log();
}
```

Controller-klasse

- Annoteren met **@RestController**
- Zelfde regels als bij niet-Reactive-stack

```
@PostMapping("/naam")
public Mono<ResponseEntity<Void>> ...
```

JDBC

Properties

```
spring.datasource.url=jdbc:postgresql://localhost:poort/naam_db
#spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.username=iii
spring.datasource.password=iiipwd
```

Connectie & DataSource

```
@Component
@PropertySource("classpath:databankconstanten.properties")
public class JDBCDataStorage implements IDataStorage {
    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    private Connection openConnectie() throws SQLException {
        return dataSource.getConnection();
    }
}
```

- Gegevens ophalen uit .properties-bestand

```
@Value("${connectiestring}")
private String connString;
```

- Connectie afsluiten

```
try (Connection conn = openConnectie()) {
    ...
} catch (Exception e) {
    ...
}
```

- Objecten ophalen
    - Schrijf hulpfuncties om objecten aan te maken die een ResultSet aanvaarden als parameter

```
@Override
public List<IObject> getObjects() throws DataExceptie {
    List<IObjects> objects;
    try(Connection conn = dataSource.getConnection()) {
        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery(SELECT_OBJECTS); //
@Value(...)
        while (resultSet.next()) {
            products.add(createObject(resultSet)); // hulpfunctie
        }
    } catch(SQLException e) {
        throw new DataExceptie(FOUT_OBJECTS);
    }
    return objects;
}
```

- Objecten ophalen aan de hand van een voorwaarde

```
@Override
public List<IObject> getObjects(int parameter) throws DataExceptie { // int kan
ook een ander type zijn
    List<IObject> objects;
    try (Connection conn = openConnectie();
            PreparedStatement stmt =
conn.prepareStatement(SELECT_OBJECTS_PARAMETER)) { // @Value(...)
        stmt.setInt(1, parameter); // of een ander type
        ResultSet rs = stmt.executeQuery();
        orders = new ArrayList<>();
        while (rs.next()) {
            objects.add(createObject(rs)); // hulpfunctie
        }
    } catch (SQLException ex) {
        throw new DataExceptie(FOUT_OBJECTS);
    }
    return objects;
}
```

- Objecten toevoegen

```
@Override
public void addObject(IObject object) throws DataExceptie {
    try (
            Connection conn = openConnectie()) {
        try {
            conn.setAutoCommit(false);
            addObject(conn, object); // hulpfunctie
            conn.commit(); // opslaan
        } catch (SQLException se) {
            conn.rollback(); // terugzetten naar vorige versie
```

```
            throw new DataExceptie(FOUT_ADD_OBJECT);
        } finally {
            conn.setAutoCommit(true);

        }
    } catch (Exception ex) {
        throw new DataExceptie(FOUT_ADD_OBJECT);
    }
}

// hulpfunctie
private void addObject(Connection conn, IObject object) throws SQLException {
        try (PreparedStatement stmt =
                    conn.prepareStatement(INSERT_OBJECT)) {
            stmt.setInt(1, object.getNumber());
            stmt.setDate(2, new java.sql.Date(object.getDate().getTime())); //
datum toevoegen
            stmt.setNull(3, java.sql.Types.DATE); //Een datum op NULL zetten; kan
met eender welk type
            stmt.executeUpdate();
        }
    }
```

- Objecten updaten

```
@Override
public void modifyObject(IObject object) throws DataExceptie {
    try (Connection conn = openConnectie(); PreparedStatement stmt =
conn.prepareStatement(UPDATE_OBJECT)) {
        stmt.setString(1, object.getParameter1());
        stmt.setString(2, object.getParameter2());
        // zo door voor elke parameter
        if (customer.getParameterThatCanBeNull() != null &&
!customer.getParameterThatCanBeNull().equals("")) {
            stmt.setString(6, object.getParameterThatCanBeNull());
        } else {
            stmt.setNull(6, java.sql.Types.VARCHAR);
        }
        // zo door voor alle parameters
        stmt.executeUpdate();

    } catch (SQLException ex) {
        throw new DataExceptie(FOUT_UPDATE_OBJECT);
    }
}
```

- Object verwijderen

```
@Override
public void deleteObject(int objectId) throws DataExceptie {
    try (Connection conn = openConnectie(); PreparedStatement stmt =
conn.prepareStatement(DELETE_OBJECT)) {
        stmt.setInt(1, objectId); // kan met elk type dat gebruikt wordt om het
object te identificeren
        stmt.executeUpdate();
    } catch (SQLException ex) {
        throw new DataExceptie(FOUT_DELETE_OBJECT);
    }
}
```