Documentation

Πέτρος-Ευστάθιος Φατούρος Π18164

----ΠΕΡΙΕΧΟΜΕΝΑ----

Το παρών documentation αποτελεί το εγχειρίδιο προγραμματιστή για την εξαμηνιαία εργασία στο μάθημα “Πληροφοριακά Συστήματα στο Διαδίκτυο”.

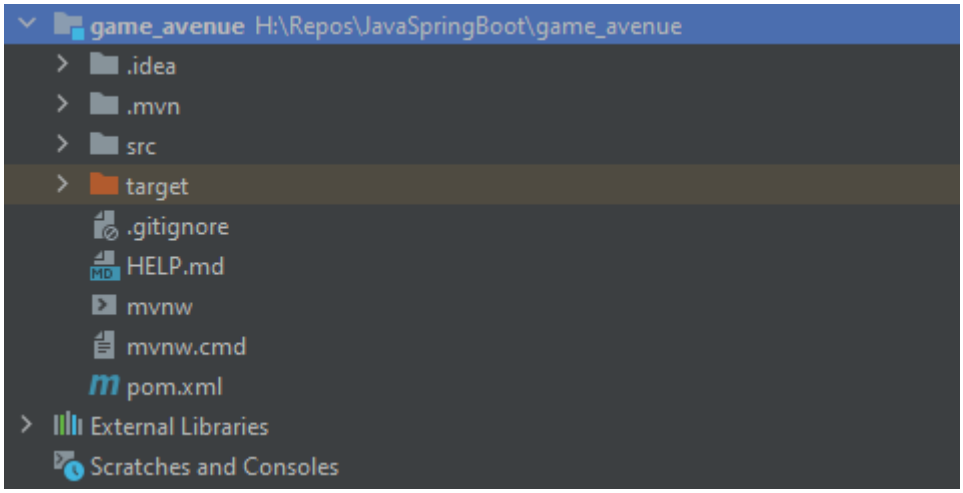
Ακολουθεί το θέμα που δηλώθηκε για την παρών εργασία.

“Θα υλοποιηθεί μια εφαρμογή που θα παρέχει πληροφορίες σχετικά με free-to-play παιχνίδια (γίνεται χρήση σχετικού API). Θα υπάρχουν δύο είδη χρηστών (απλός χρήστης και διαχειριστής). Κατά την εγγραφή θα ζητείται από το χρήστη, μεταξύ άλλων, email και τηλέφωνο τα οποία θα επαληθεύονται με τη χρήση σχετικών API. Θα υπάρχει η δυνατότητα για real time μετάφραση της εφαρμογής με τη χρήση του Google Translate API. Κάθε χρήστης θα έχει μία λίστα στην οποία θα μπορεί να αποθηκεύει παιχνίδια. Θα υπάρχει εξεζητημένη αναζήτηση παιχνιδιών. Συγκεκριμένα θα δίνεται η δυνατότητα για αναζήτηση παιχνιδιού με βάση: α) title, β) platform, γ) genre, δ) όλα τα προηγούμενα μαζί και επιπλέον θα υπάρχει και η δυνατότητα για ταξινόμηση των αποτελεσμάτων κατά ημερομηνία κυκλοφορίας, αλφαβητική σειρά ή συνάφεια (relevance), ε) πολλαπλά tags.”

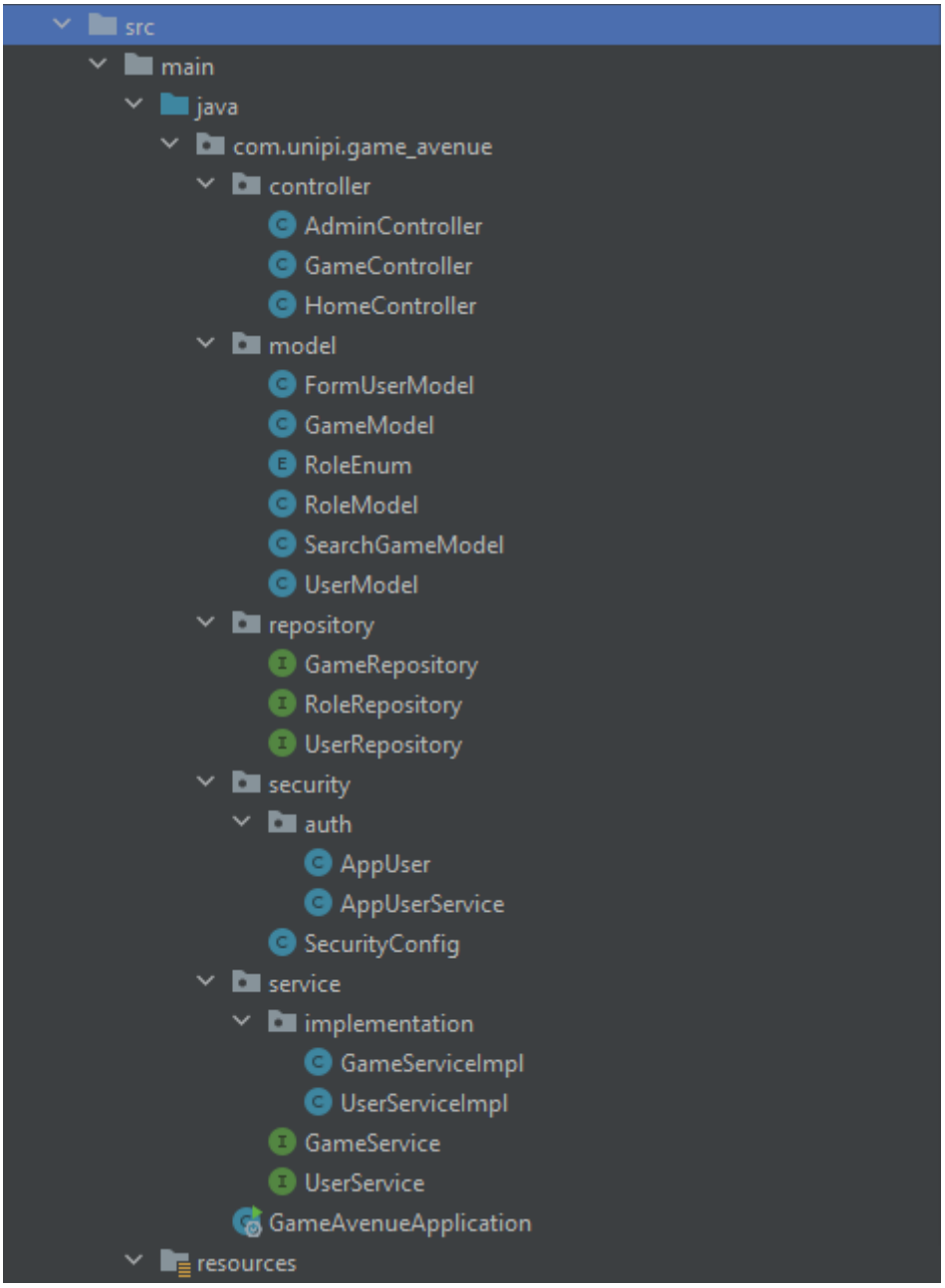
Δομή του Project και περιγραφή αρχείου “pom.xml”

Η εργασία έχει γραφθεί σε Java Spring Boot σε περιβάλλον IntelliJ IDEA.

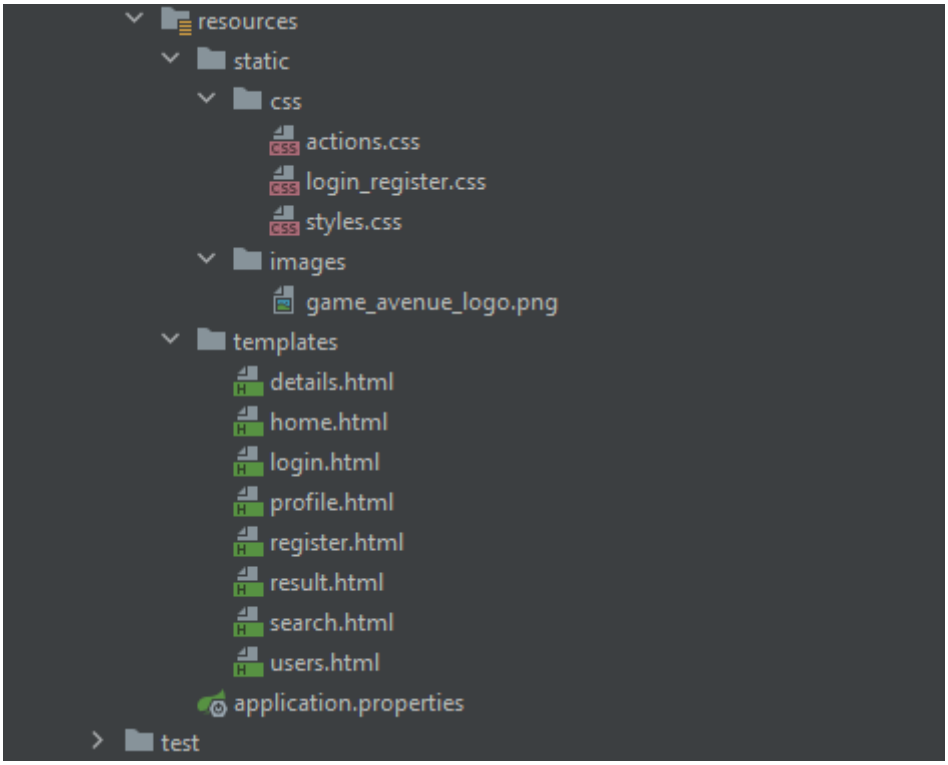
Ακολουθούν ενδεικτικά screenshots σχετικά με τη δομή του project.



Το όνομα του project είναι “game_avenue”. Τα αρχεία που είναι ουσιαστικής σημασίας, τα οποία και θα αναλυθούν, είναι αυτά που βρίσκονται κάτω από το φάκελο “src” και το αρχείο “pom.xml”.



Όλα τα packages, classes, interfaces, enum που βρίσκονται κάτω από το φάκελο “java” θα αναλυθούν παρακάτω στο documentation.



Αντίστοιχα όλα τα αρχεία που βρίσκονται κάτω από το φάκελο “resources” θα αναλυθούν επίσης παρακάτω στο documentation.

Ακολουθεί επεξήγηση του αρχείου “pom.xml”.

Για την παραγωγή του βασικού project έγινε χρήση του Java Initializr (https://start.spring.io).

Ακολουθούν σχετικά screenshots.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.7</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.unipi</groupId>
    <artifactId>game_avenue</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>game_avenue</name>
    <description>Game Avenue web application</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
```

Παραπάνω εμφανίζονται βασικές πληροφορίες σχετικά με το project όπως τα μεταδεδομένα του και η έκδοση Spring Boot.

Τα dependencies που χρησιμοποιήθηκαν είναι τα εξής:

- Spring Data JPA (δημιουργία των Entities και Repositories)
- Spring Security (authentication και authorization της εφαρμογής)
- Thymeleaf (template engine)
- Validation (επικύρωση πεδίων και αντικειμένων)
- Spring Web (δημιουργία web εφαρμογής)
- PostgreSQL Driver (επικοινωνία της εφαρμογής με τη βάση)
- Lombok (μείωση boilerplate code)
- OkHttp (κλήσεις στα API που χρησιμοποιούνται)
- JSON in Java (αποκωδικοποίηση των JSON responses των API)

Ακολουθούν screenshots με τα dependencies.

```xml
<dependencies>

    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Spring Security -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <!-- Thymeleaf -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- Validation -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
    </dependency>
```

```xml
<!-- PostgreSQL Driver -->
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- Lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

<!-- OkHttp -->
<dependency>
    <groupId>com.squareup.okhttp</groupId>
    <artifactId>okhttp</artifactId>
    <version>2.7.5</version>
</dependency>

<!-- JSON in Java-->
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20220320</version>
</dependency>

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

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

Τέλος, ακολουθεί screenshot σχετικά με το build του project.

```
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Έχοντας αναλύσει τη δομή του project, το αρχείο “pom.xml” και το πως παράχθηκε το βασικό project, ακολουθεί επεξήγηση και ανάλυση των βασικών λειτουργιών και της αλγοριθμικής διαδικασίας που έχουν αναπτυχθεί στην εργασία, με βάση το θέμα που δόθηκε στην αρχή του documentation.

Σχετικά με τη Βάση Δεδομένων

Η βάση δεδομένων που χρησιμοποιήθηκε είναι PostgreSQL. Δημιουργήθηκε βάση με όνομα “game_avenue_db” με τα εξής tables (τα οποία δημιουργήθηκαν μέσω Spring Data JPA):

- games (το table αυτό αντιστοιχεί στη κλάση GameModel που βρίσκεται στο πακέτο model. Τα πεδία του αντιστοιχούν στα αντίστοιχα πεδία του API που χρησιμοποιήθηκε: https://rapidapi.com/digiwalls/api/free-to-play-games-database. Ακολουθεί ενδεικτικό screenshot)

```
26 usages
@Entity
@Table(name = "games")
@NoArgsConstructor
@AllArgsConstructor
@Getter
/JpaDataSourceORMInspection/
public class GameModel {

    @Id
    private Long id;

    @Column(length = 50, nullable = false)
    private String title;

    @Column(length = 100, nullable = false)
    private String thumbnail;

    @Lob
    @Type(type = "org.hibernate.type.TextType")
    @Column(nullable = false)
    private String short_description;

    @Column(length = 100, nullable = false)
    private String game_url;

    @Column(length = 30, nullable = false)
    private String genre;

    @Column(length = 30, nullable = false)
    private String platform;

    @Column(length = 100, nullable = false)
    private String publisher;

    @Column(length = 100, nullable = false)
    private String developer;

    @Column(length = 20, nullable = false)
    private String release_date;
}
```

- roles (το table αυτό αντιστοιχεί στη κλάση RoleModel που βρίσκεται στο πακέτο model. Αποτελείται από ένα χαρακτηριστικό πεδίο Id και το όνομα του ρόλου το οποίο αντλεί από το enum RoleEnum που βρίσκεται επίσης στο πακέτο model. Κάθε χρήστης μπορεί να έχει παραπάνω από έναν ρόλο. Ακολουθούν ενδεικτικά screenshots)

```
7 usages
@Entity
@Table(name = "roles")
@NoArgsConstructor
@Getter
/JpaDataSourceORMInspection/
public class RoleModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Enumerated(EnumType.STRING)
    @Column(length = 20, nullable = false)
    private RoleEnum name;
}
```

```
public enum RoleEnum {

    1 usage
    ROLE_USER,
    1 usage
    ROLE_ADMIN
}
```

- users (το table αυτό αντιστοιχεί στη κλάση UserModel που βρίσκεται στο πακέτο model. Τα πεδία του αντιστοιχούν σε πληροφορίες σχετικές με το χρήστη: όνομα χρήστη, email, τηλέφωνο, κωδικός. Το πεδίο Id είναι το όνομα χρήστη. Επιπλέον, κάθε χρήστης έχει μία λίστα με παιχνίδια και ένα set με ρόλους. Η πρώτη σχέση εκφράζεται από το table “user_games” και η δεύτερη από το table “user_roles. Ακολουθεί ενδεικτικό screenshot)

```
15 usages
@Entity
@Table(name = "users")
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
/JpaDataSourceORMInspection/
public class UserModel {

    @Id
    @Column(length = 20)
    private String username;

    @Column(length = 320, nullable = false, unique = true)
    private String email;

    @Column(name = "phone_number", length = 15, nullable = false, unique = true)
    private String phoneNumber;

    @Column(length = 60, nullable = false)
    private String password;

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<RoleModel> roles = new HashSet<>();

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "user_games",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "game_id"))
    private List<GameModel> games = new ArrayList<>();
}
```

- user_games (βλέπετε πληροφορίες σχετικά με table “users”)
- user_roles (βλέπετε πληροφορίες σχετικά με table “users”)

Τα queries στη βάση γίνονται μέσω των Jpa Repositories που αντιστοιχούν στα interfaces βρίσκονται στο πακέτο “repository”. Ακολουθεί ενδεικτικό screenshot του UserRepository το οποίο είναι παρομοιότυπο με τα άλλα δύο αντίστοιχα interfaces.

```
6 usages
public interface UserRepository extends JpaRepository<UserModel, String> {
    1 usage
    boolean existsByEmail(String email);

    1 usage
    boolean existsByPhoneNumber(String phoneNumber);
}
```

Configurations σχετικά με τη βάση δεδομένων υπάρχουν στο αρχείο “application.properties”. Ακολουθεί σχετικό screenshot.

```
# Database Configurations
spring.datasource.url=jdbc:postgresql://localhost:5432/game_avenue_db
spring.datasource.username=postgres
spring.datasource.password=admin
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true
```

Για να εκτελέσετε την εφαρμογή στο δικό σας υπολογιστή, θα πρέπει να προσαρμόσετε και τα αντίστοιχα πεδία (url, username, password). Επίσης, την πρώτη φορά της εκτέλεσης της εφαρμογής, θέτετε το πεδίο “ddl-auto” με τιμή “create” αντί για “none”, ώστε να δημιουργηθούν τα tables της βάσης.

Resources του Project

Με αφορμή το αρχείο “application.properties” ακολουθεί ανάλυση του φακέλου “resources” του project.

Όλα τα αρχεία html βρίσκονται στο φάκελο “templates”.

- **login.html**: αποτελεί τη σελίδα από την οποία ξεκινάει η εφαρμογή. Ζητάει από το χρήστη username και password ώστε να συνδεθεί.

- **register.html**: αποτελεί τη σελίδα από την οποία πραγματοποιείται η εγγραφή του χρήστη στην εφαρμογή. Ζητάει από το χρήστη username, email, phone number, password και επαλήθευση password. Κατά την εγγραφή ο χρήστης αποκτά αυτόματα το ρόλο "ROLE_USER". Την ίδια σελίδα χρησιμοποιεί και ο διαχειριστής όταν επιθυμεί να καταχωρήσει έναν χρήστη στο σύστημα. Ο διαχειριστής έχει την επιλογή να αναθέσει οποιοδήποτε ρόλο στο χρήστη (ακόμα και τους δύο ρόλους).

- **home.html**: αποτελεί τη σελίδα στην οποία ανακατευθύνεται ο χρήστης μετά από ένα επιτυχημένο login. Ανάλογα με το ρόλο (ή ρόλους) του χρήστη, εμφανίζει και τις αντίστοιχες δράσεις. Ο απλός χρήστης έχει τις δυνατότητες να αναζητήσει παιχνίδια, να καταχωρήσει παιχνίδια στη λίστα του και να δει την εν λόγο λίστα με τα αποθηκευμένα παιχνίδια. Ο διαχειριστής έχει τις δυνατότητες να κάνει update τη βάση δεδομένων, να καταχωρήσει ή να διαγράψει χρήστη από τη βάση δεδομένων και να δει τη λίστα με τους υπάρχων χρήστες της εφαρμογής.

- **search.html**: αποτελεί τη σελίδα από την οποία ο χρήστης αναζητά παιχνίδια (βασική ή προχωρημένη αναζήτηση). Στη βασική αναζήτηση ζητάει από το χρήστη τίτλο και προαιρετικά κατηγορία παιχνιδιού, πλατφόρμα και τον τρόπο ταξινόμησης των αποτελεσμάτων. Στη προχωρημένη αναζήτηση ζητάει από το χρήστη τίτλο, κατηγορία παιχνιδιού (ή κατηγορίες) και προαιρετικά.

- **result.html**: αποτελεί τη σελίδα από την οποία ο χρήστης βλέπει τα αποτελέσματα της αναζήτησής του. Για κάθε παιχνίδι που προβάλλεται υπάρχει το κουμπί "Details" το οποίο ανακατευθύνει το χρήστη σε άλλη σελίδα που περιέχει περισσότερες πληροφορίες και λεπτομέρειες σχετικά με το παιχνίδι.

- **details.html**: αποτελεί τη σελίδα από την οποία ο χρήστης βλέπει πληροφορίες σχετικά με το επιλεγμένο παιχνίδι. Υπάρχει αντίστοιχο κουμπί με το οποίο ο χρήστης μπορεί να προσθέσει στη λίστα του το συγκεκριμένο παιχνίδι (ή να το αφαιρέσει αν ήδη υπάρχει).

- **profile.html**: αποτελεί τη σελίδα από την οποία ο χρήστης βλέπει την προσωπική του λίστα με τα αποθηκευμένα παιχνίδια.

- **users.html**: αποτελεί τη σελίδα από την οποία ο διαχειριστής βλέπει τη λίστα με τους υπάρχων χρήστες της εφαρμογής. Έχει τη δυνατότητα να διαγράψει οποιονδήποτε χρήστη πέρα από τον εαυτό του.

Στο φάκελο “css” βρίσκονται τα αντίστοιχα css αρχεία που χρησιμοποιήθηκαν για το styling των σελίδων html. Για το styling επιπλέον χρησιμοποιήθηκε Bootstrap (CDN via jsDelivr).

https://getbootstrap.com/docs/5.2/getting-started/download/#cdn-via-jsdelivr

Στο φάκελο “images” βρίσκονται το logo της εφαρμογής. Δημιουργήθηκε μέσω https://www.freelogodesign.org.

Spring Security

Το authentication και το authorization της εφαρμογής γίνεται χρησιμοποιώντας spring security.

Τα στοιχεία του χρήστη κατά τη διαδικασία του login αντιστοιχούν στη κλάση AppUser του πακέτου auth, η οποία κάνει implement τη διεπαφή UserDetails.

Υπάρχει επίσης η αντίστοιχη service κλάση AppUserService, η οποία κάνει implement τη διεπαφή UserDetailsService.

Τέλος, η υπάρχει η configuration κλάση SecurityConfig η οποία κάνει extend τη κλάση WebSecurityConfigurerAdapter. Από τη συγκεκριμένη κλάση αξίζει να δοθεί περισσότερη σημασία σε μια συγκεκριμένη μέθοδο. Ακολουθεί screenshot της μεθόδου.

```
// Configure http security
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
            .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
                .antMatchers( ...antPatterns: "/register", "/css/*", "/images/*").permitAll()
                .anyRequest().authenticated()
            .and() HttpSecurity
            .formLogin() FormLoginConfigurer<HttpSecurity>
                .loginPage("/login")
                .defaultSuccessUrl( defaultSuccessUrl: "/home", alwaysUse: true)
                .permitAll()
            .and() HttpSecurity
            .logout() LogoutConfigurer<HttpSecurity>
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login")
                .invalidateHttpSession(true)
                .deleteCookies( ...cookieNamesToClear: "JSESSIONID")
                .permitAll();
}
```

Με λίγα λόγια, η παραπάνω μέθοδος παραχωρεί πρόσβαση στα στατικά αρχεία της εφαρμογής που βρίσκονται στους φακέλους “css” και “images”, καθώς και δίνει πρόσβαση στο χρήστη στο path “/register” χωρίς να χρειάζεται ο χρήστης να είναι authenticated. Για να αποκτήσει πρόσβαση σε όλα τα υπόλοιπα paths πρέπει να είναι authenticated. Αυτό γίνεται μέσω της σελίδας login, η οποία πάντα μετά από μια επιτυχή σύνδεση ανακατευθύνει το χρήστη στο route “/home”. Τέλος, όταν ο χρήστης αποσυνδέεται από την εφαρμογή, ανακατευθύνεται αυτόματα στη σελίδα login και διαγράφεται το ήδη υπάρχων session και το αντίστοιχο cookie.

Controllers and Routes

Στο πακέτο “controllers” υπάρχουν 3 κλάσεις, οι οποίες αναλύονται παρακάτω.

HomeController

- GET: / (εμφανίζει τη σελίδα home.html)
- GET: /home (εμφανίζει τη σελίδα home.html)
- GET: /login (εμφανίζει τη σελίδα login.html)
- GET: /register (εμφανίζει τη σελίδα register.html)
- POST: /register (πραγματοποιεί την εγγραφή του χρήστη)

AdminController (πρέπει να έχει ρόλο διαχειριστή για να έχει πρόσβαση)

- GET: /users (εμφανίζει τη σελίδα users.html)
- GET: /delete/{username} (διαγράφει τον επιλεγμένο χρήστη και ανακατευθύνει στη σελίδα users.html)
- GET: /update (φορτώνει τη βάση δεδομένων με όλα τα τρέχων παιχνίδια που είναι αποθηκευμένα στο API)

GameController

- GET: /search/{search_type} (εμφανίζει τη σελίδα search.html η οποία τροποποιείται μέσω Thymeleaf ανάλογα με το αν ο χρήστης έχει επιλέξει βασική ή προχωρημένη αναζήτηση παιχνιδιών)
- POST: /search (ανακατευθύνει το χρήστη στη σελίδα result.html όπου εμφανίζονται τα αποτελέσματα της αναζήτησης του)
- GET: /search_results (εμφανίζει τη σελίδα result.html)
- GET: /details/{game_id} (εμφανίζει τη σελίδα details.html όπου ο χρήστης μπορεί να δει πληροφορίες σχετικά με το επιλεγμένο παιχνίδι)
- GET: /remove/{game_id} (αφαιρεί το επιλεγμένο παιχνίδι από τη λίστα του χρήστη και ανακατευθύνει στη σελίδα details.html)

- GET: /profile (εμφανίζει τη σελίδα profile.html)

Για τη μεταφορά δεδομένων από τις φόρμες στους controllers χρησιμοποιήθηκαν οι κλάσεις FormUserModel (για τη μεταφορά δεδομένων χρηστών) και SearchGameModel (για τη μεταφορά δεδομένων παιχνιδιών)

Service

Οι περισσότεροι από τους προαναφερθείσας controllers χρησιμοποιούν κάποια μέθοδο από κάποιο service. Στην εφαρμογή αυτή υπάρχει ένα service για τους χρήστες (UserService) και ένα αντίστοιχο service για τα παιχνίδια (GameService). Οι μέθοδοι στις διεπαφές αυτές υλοποιούνται από τις αντίστοιχες κλάσεις (UserServiceIml και GameServiceIml).

Ακολουθούν ενδεικτικά screenshots των interfaces.

```
2 usages  1 implementation
public interface UserService {

    1 usage  1 implementation
    BindingResult registerUser(FormUserModel user, BindingResult result) throws IOException;

    1 usage  1 implementation
    Boolean verifyEmail(String email) throws IOException;

    1 usage  1 implementation
    Boolean verifyPhoneNumber(String phone_number) throws IOException;

    1 usage  1 implementation
    List<UserModel> getAllUsers();

    1 usage  1 implementation
    void deleteUser(String username);

    1 usage  1 implementation
    UserModel getCurrentUser();
}
```

```java
2 usages  1 implementation
public interface GameService {

    3 usages  1 implementation
    List<GameModel> callApi(String url) throws IOException;

    1 usage  1 implementation
    List<String> getAllTags();

    1 usage  1 implementation
    List<GameModel> searchGames(SearchGameModel search_data) throws IOException;

    1 usage  1 implementation
    void updateDatabase() throws IOException;

    1 usage  1 implementation
    GameModel getGame(Long game_id) throws IOException;

    1 usage  1 implementation
    void saveGameToList(Long game_id);

    1 usage  1 implementation
    List<GameModel> getUserGamesList();

    1 usage  1 implementation
    void deleteGameFromList(Long game_id);
}
```

Το κύριο API που χρησιμοποιεί η εφαρμογή είναι το https://rapidapi.com/digiwalls/api/free-to-play-games-database το οποίο αποτελεί μία βάση από free-to-play παιχνίδια.

Για τη βασική και προχωρημένη αναζήτηση χρησιμοποιείται η μέθοδος callApi του GameService. Ακολουθούν ενδεικτικά screenshots του κώδικα.

```
3 usages
@Override
public List<GameModel> callApi(String url_api) throws IOException {
    OkHttpClient client = new OkHttpClient();

    // Request
    Request request = new Request.Builder()
            .url(url_api)
            .get()
            .addHeader( name: "X-RapidAPI-Host",  value: "free-to-play-games-database.p.rapidapi.com")
            .addHeader( name: "X-RapidAPI-Key",  value: "85d7bede2fmsh905c6165a0954bbp1b65e5jsn2e7b36cdd213")
            .build();

    // Get Response
    Response response = client.newCall(request).execute();
```

Κάθε φορά η μέθοδος δέχεται το url ως παράμετρο. Η ίδια μέθοδος χρησιμοποιείται στη βασική και στην προχωρημένη αναζήτηση.

Αφού λάβει η εφαρμογή το response από το API, το αποκωδικοποιεί και επιστρέφει μία λίστα με GameModel αντικείμενα (η συνέχεια του κώδικα στο επόμενο screenshot).

```
    // Get Response
    Response response = client.newCall(request).execute();

    // Gather games in a list
    List<GameModel> games = new ArrayList<>();

    String responseBody = response.body().string();

    // The games from the response are in JSON format
    JSONArray json_games;

    try{
        json_games = new JSONArray(responseBody);
    }
    catch (JSONException e){
        json_games = new JSONArray();
    }

    // Convert the JSON formatted games into GameModel objects
    for (int i = 0; i < json_games.length(); i++) {

        JSONObject game = json_games.getJSONObject(i);
        games.add(new GameModel(
                game.getLong( key: "id"),
                game.getString( key: "title"),
                game.getString( key: "thumbnail"),
                game.getString( key: "short_description"),
                game.getString( key: "game_url"),
                game.getString( key: "genre"),
                game.getString( key: "platform"),
                game.getString( key: "publisher"),
                game.getString( key: "developer"),
                game.getString( key: "release_date")
        ));
    }

    return games;
}
```

Ο έλεγχος εγκυρότητας του email και του τηλεφώνου γίνονται από τις μεθόδους VerifyEmail και VerifyPhoneNumber του UserService. Ακολουθούν ενδεικτικά screenshots του κώδικα των μεθόδων.

```java
// Verify user's email using API
// https://rapidapi.com/mr_admin/api/email-verifier/
1 usage
@Override
public Boolean verifyEmail(String email) throws IOException {

    // Prepare email
    email = email.replace( target: "@",  replacement: "%40");

    OkHttpClient client = new OkHttpClient();

    Request request = new Request.Builder()
            .url("https://mailcheck.p.rapidapi.com/?domain=" + email)
            .get()
            .addHeader( name: "X-RapidAPI-Key",  value: "85d7bede2fmsh905c61b5a0954bbp1b65e5jsn2e7b36cdd213")
            .addHeader( name: "X-RapidAPI-Host",  value: "mailcheck.p.rapidapi.com")
            .build();

    Response response = client.newCall(request).execute();

    String responseBody = response.body().string();

    JSONObject json_response = new JSONObject(responseBody);

    boolean valid = json_response.getBoolean( key: "valid");
    boolean block = json_response.getBoolean( key: "block");
    boolean disposable = json_response.getBoolean( key: "disposable");

    return valid && !block && !disposable;
}
```

Σημείωση: χρησιμοποιήθηκε διαφορετικό API για τον έλεγχο εγκυρότητας του email. Ο λόγος είναι το αρχικό ήταν επί πληρωμή (PAID) ενώ αυτό που τελικά χρησιμοποιήθηκε στην εφαμογή ήταν δωρεάν (FREEMIUM).

Χρησιμοποιήθηκε το API https://rapidapi.com/Top-Rated/api/e-mail-check-invalid-or-disposable-domain.

```java
// Verify user's phone number using API
// https://rapidapi.com/Veriphone/api/veriphone/
1 usage
@Override
public Boolean verifyPhoneNumber(String phone_number) throws IOException {

    OkHttpClient client = new OkHttpClient();

    Request request = new Request.Builder()
            .url("https://veriphone.p.rapidapi.com/verify?phone=" + phone_number)
            .get()
            .addHeader( name: "X-RapidAPI-Key",  value: "85d7bede2fmsh905c6165a0954bbp1b65e5jsn2e7b36cdd213")
            .addHeader( name: "X-RapidAPI-Host",  value: "veriphone.p.rapidapi.com")
            .build();

    Response response = client.newCall(request).execute();

    String responseBody = response.body().string();

    JSONObject json_response = new JSONObject(responseBody);

    return json_response.getBoolean( key: "phone_valid");
}
```

Για τον έλεγχο εγκυρότητας του τηλεφώνου, χρησιμοποιήθηκε το API https://rapidapi.com/Veriphone/api/veriphone.

Και στις δύο περιπτώσεις (email και τηλέφωνο), οι μέθοδοι κάνουν κλήση στο API, παίρνουν ένα response, το αποκωδικοποιούν και επιστρέφουν αναλόγος true ή false (έγκυρο ή όχι έγκυρο).

Σημείωση: το Google Transalte API έχει ενσωματωθεί μέσα στις σελίδες html.

Χρησιμοποιείται το παρακάτω tag το οποίο περιέχει ένα select με τις διαθέσιμες προς μετάφραση γλώσσες. Για την πραγμοτοποίηση της μετάφρασης χρησιμοποιείται αντίστοιχο script. Ακολυθούν screenshots.

```html
<div id="google_translate_element"></div>
```

```html
<script type="text/javascript">
    function googleTranslateElementInit() {
        new google.translate.TranslateElement({pageLanguage: 'en'}, 'google_translate_element');
    }
</script>
<script type="text/javascript" src="//translate.google.com/translate_a/element.js?cb=googleTranslateElementInit"></script>
```

Σχετικά με το παραδοτέο

Το παραδοτέο αποτελείται από:

- Το IntelliJ IDEA project που βρίσκεται μέσα στο φάκελο game_avenue (υπάρχουν επιπλέον επεξηγηματικά σχόλια μέσα στο κώδικα)
- Το παρών documentation (Documentation.pdf)
- Το user manual το οποίο αποτελεί παράδειγμα εκτέλεσης της εφαρμογής (User_Manual.pdf)
- Τα tables της βάσης σε μορφή αρχείου csv ακριβώς όπως προκύπτουν μετά το παράδειγμα εκτέλεσης στο user manual. Βρίσκονται μέσα στο φάκελο tables.
- Οι εντολές insert που χρησιμοποιήθηκαν στο παράδειγμα εκτέλεσης (insert.sql)