Desktop aplikacija za igru Sudoku

Madžarević, Karlo

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni prijediplomski studij Računarstvo

DESKTOP APLIKACIJA ZA IGRU SUDOKU

Završni rad

Karlo Madžarević

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Karlo Madžarević
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R 4390, 08.10.2020.
JMBAG:	0165082115
Mentor:	Marina Peko, dipl. ing. el.
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Ivana Hartmann Tolić
Član Povjerenstva 1:	Marina Peko, dipl. ing. el.
Član Povjerenstva 2:	doc. dr. sc. Krešimir Romić
Naslov završnog rada:	Desktop aplikacija za igru Sudoku
Znanstvena grana završnog rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Kreirati desktop aplikaciju za igru Sudoku. Korisnik/igrač treba imati mogućnost samostalnog unosa polja sudoku elemenata ili igrati predefinirane igre. Veličina polja je varijabilna te podložna promjeni. Igračima se pamte ostvareni bodovi i prikazuju u rang listi. Detalji aplikacija će biti dogovoreni s mentorom nakon izbora teme.

Datum ocjene pismenog dijela završnog rada od strane mentora:	23.09.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Vrlo dobar (4)
Datum obrane završnog rada:	30.09.2024.
Ocjena usmenog dijela završnog rada (obrane):	Izvrstan (5)
Ukupna ocjena završnog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	03.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 03.10.2024.

Ime i prezime Pristupnika:	Karlo Madžarević
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4390, 08.10.2020.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Desktop aplikacija za igru Sudoku**

izrađen pod vodstvom mentora Marina Peko, dipl. ing. el.

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Tema ovog završnog rada je izrada aplikacije za računala (engl. *Desktop application*) za igranje logičke zagonetke Sudoku. Cilj ove aplikacije je omogućiti korisniku odabir različitih težina koje korisnik rješava te zapis i prikaz vremena u kojem je korisnik riješio zadanu zagonetku. Pazeći na kriterije pojedine zagonetke, korisnik rješava Sudoku zagonetku na način da upisuje brojeve u prazna, za to predviđena polja. Bit će izrađeno jednostavno sučelje s funkcijama potrebnim kako bi igrač mogao postaviti svoj nadimak kako bi se mogla voditi evidencija o vremenu potrebnom za rješavanje te zagonetke. Korisnik unosi brojeve tako da se brojevi ne ponavljaju ni u istom retku ni u istom stupcu i ni u istom kvadratu. Aplikacija je napravljena u Java programskom jeziku i namijenjena je za uređaje s instaliranim JVM-om. JVM je kratica za *Java Virtual Machine,* softverski sloj koji omogućuje izvršavanje Java programa prevođenjem Java *bytecode*-a u računalni kod za pojedinu platformu. Pravila i povijest igre Sudoku opisani su u 2. poglavlju. Aplikacija je pisana u Intellij razvojnom okruženju, Java verzije 17, detaljnije objašnjeno u poglavlju 3. Razvoj aplikacije objašnjen je u poglavlju 4.

1.1. Zadatak završnog rada

Zadatak rada bio je napraviti aplikaciju za igranje igre Sudoku. Aplikacija treba poštivati sva pravila igre Sudoku te omogućiti prijavu korisnika kako bi se mjerilo vrijeme potrebno za rješavanje zagonetke. Aplikacija omogućuje korisniku da se ne prijavljuje, već da nastavi kao gost.

2. PREGLED POSTOJEĆIH RJEŠENJA

Igra Sudoku matematička je zagonetka čije se rješavanje temelji na logičkom zaključivanju. Porijeklom je iz Japana, gdje je prethodnik Sudokua igra magični kvadrat, no Sudoku u ovom obliku postaje od 2005. poznat u cijelom svijetu. [1]

Igra se tako da se napravi kvadratno polje od 9 × 9 polja, koji se sastoji od 9 manjih kvadrata dimenzije 3 × 3. Postoje i druge inačice Sudokua, ali najpopularnija je 9 × 9, stoga su pravila objašnjena na temelju te inačice. Igrač dobiva zadano 9 × 9 polje s brojevima na slučajnim mjestima te igrač popunjava tako da na svako mjesto ide odgovarajući broj od 1 do 9 s uvjetima da se taj broj ne smije pojavljivati nigdje drugo u tom stupcu i retku i ne smije se pojavljivati u tom kvadratu. Svaki kvadrat mora sadržavati sve brojeve od 1 do 9. [2]

2.1. Postojeće aplikacije za računala

Sudoku aplikacije postale su jedan od načina za igranje, uz mobilne i internetske aplikacije (engl. *Web application)*. Jedno od popularnijih rješenja, prikazano slikom 2.1. je Microsoft Sudoku koju je moguće preuzeti na *Microsoft Store*-u, aplikacija prilagođena entuzijastima i povremenim korisnicima uz razne značajke, kao što su različite razine težine, dnevni izazovi kako bi se korisnik svakodnevno vraćao koristiti aplikaciju, nudi savjete, poništenje posljednjeg koraka te mogućnost

Slika 2.1. Slika igre Microsoft Sudoku na Microsoft Store-u

automatske provjere. Zatim jedno od rješenja je Sudoku Classic!, isto dostupna na *Microsoft Store*-u koja korisniku, uz slične mogućnosti kao Microsoft Sudoku, omogućuje korisniku unos podataka pomoću pametne olovke, tipkovnice, dodira ili pokazivača. Dozvoljava korisniku da si

personalizira pozadinu i zvučne efekte. Sudoku – Pro isto je aplikacija dostupna na *Microsoft Store*-u kao posebnu značajku ima 5 različitih tipova igre, od kojih je najpopularniji HyperSudoku koji dodaje dodatan sloj kompleksnosti tako što se dodaju 4 dodatna 3 × 3 kvadrata koje se preklapaju sa sredinom 9 × 9 polja koja isto trebaju imati isto unikatne brojeve od 1 do 9.

2.1.1. Mobilna i Web rješenja

Mobilna aplikacija Sudoku.com aplikacija je koja se može koristiti na svim popularnim mobilnim platformama. Korisniku omogućuje odabir jedne od 6 težinskih razina, nudi savjete i alat za provjeru grešaka za korisnike kada su zapeli ili napravili pogrešan potez. Unutar aplikacije se isto nalaze dnevni izazovi, statistika te prilagodljivu ploču za igranje igre. *Web* aplikacija WebSudoku.com nudi široku kolekciju Sudoku puzli kroz 4 težine. Zatim korisnik može direktno poslati igre na printanje ili igrati protiv drugih korisnika u stvarnom vremenu. U aplikaciji su prisutni i brojač vremena i statistika kako bi korisnik mogao lakše pratiti svoje performanse.

3. PROGRAMSKI JEZIK JAVA I INTEGRIRANO RAZVOJNO OKRUŽENJE INTELLIJ

Kako bi se mogla izraditi funkcionalna i jednostavna aplikacija, potrebno je odabrati prave alate za to. U daljnjem tekstu objašnjene su korištene tehnologije i alati za izradu aplikacije.

3.1. Programski jezik Java

Java je objektno orijentirani programski jezik razvijen u tvrtki Sun Microsystems, a prva verzija izdana je u studenom 1995. Prednost u odnosu na ostale tadašnje programske jezike je ta da se programi pisani u Java programskom jeziku mogu izvoditi na svim operacijskim sustavima za koje postoji JVM. JVM je ujedno i interpreter za programski jezik Java što omogućuje da se napisani kod ne prevodi u strojni jezik, već se sastavlja u bajtkod (engl. *Bytecode)*. Danas je jedan od najrasprostranjenijih programskih jezika, pruža visok stupanj sigurnosti i pouzdanosti zbog svog zatvorenog okoliša. Za potrebe izrade Sudoku desktop aplikacije koristit će se verzija Java 17. [3]

3.2. Integrirano razvojno okruženje Intellij

Integrirano razvojno okruženje (engl. *Integrated development environment*), kratica *IDE*, Intellij razvijeno je od tvrtke JetBrains za razvoj softvera u programskim jezicima Java, Kotlin, Scala i Groovy. Prva verzija izdana je u siječnju 2001. i time je postao jedan od prvih razvojnih okruženja za programski jezik Java. Neke od stavki koje su donijele uspjeh razvojnom okruženju Intellij: pametno i učinkovito uređivanje koda pomoću automatskog dovršavanja koda, analize, navigacije i refaktoriranja, integracija s alatima za verzioniranje, podrška alatima za izgradnju (eng. *Build*) projekta, automatsko upravljanje ovisnostima u projektu (eng. *Dependency management*), podrška za različite testne frameworke kao što su Junit i TestNG, podrška za dodatke i proširenja te podrška za više operacijskih sustava kao što su Windows, Linux i macOS. Za potrebe izrade Sudoku desktop aplikacije koristit će se verzija Build #IU-241.19072.14.

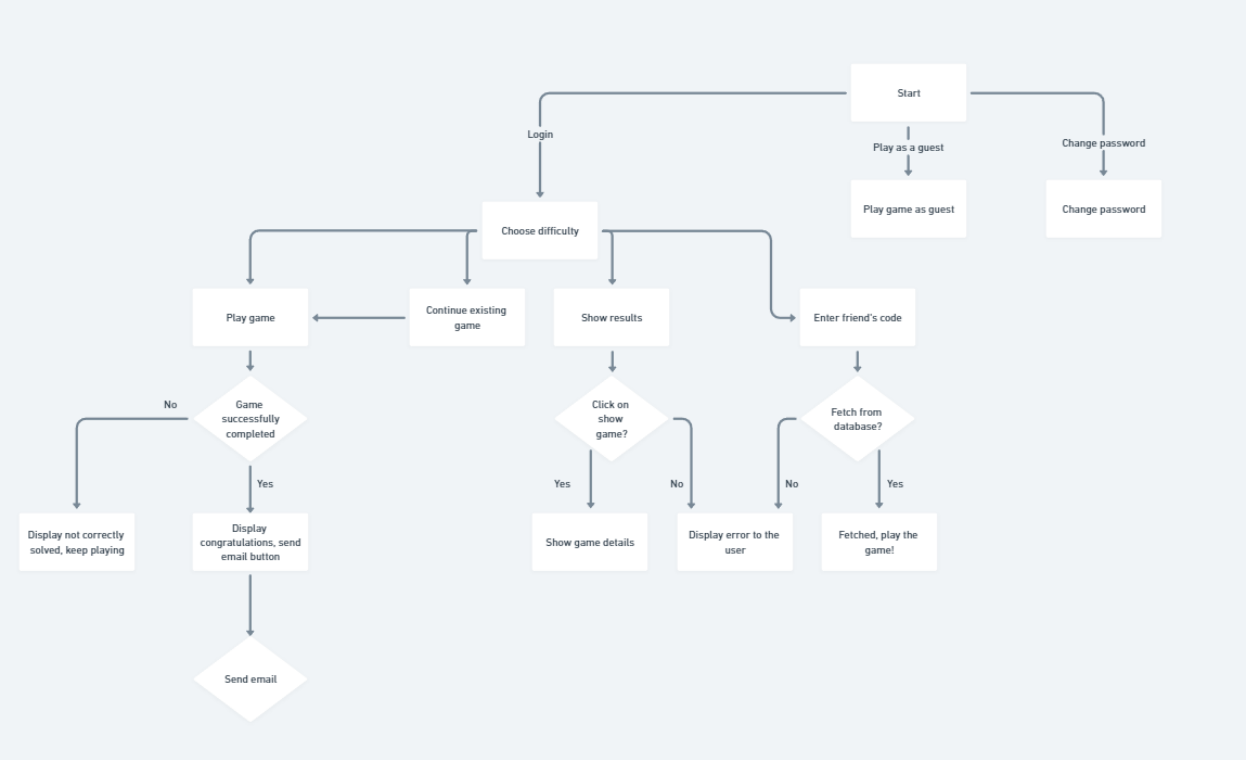
3.3. Firebase Realtime baza podataka

Firebase [4] Realtime Database [5] je NoSql [6] baza podataka koju Google nudi kao dio svog Firebase ekosustava. Koristi se za pohranu i sinkronizaciju podataka u stvarnom vremenu između klijenata i servera. Podatci se automatski sinkroniziraju na svim povezanim uređajima, koristi JSON (engl. *JavaScript Object Notation*) [7] strukturu za pohranu podataka. Nije potrebno podizati lokalnu bazu jer se sve nalazi u oblaku (engl. *Cloud*).

4. KORISNIČKI IZGLED APLIKACIJE

Zahtjevi se definiraju prema određenim značajkama postavljenih prije kreiranja aplikacije te je potrebno napraviti plan i model te ih se potrebno pridržavati u svrhu boljeg krajnjeg rezultata. U tu svrhu izrađen je dijagram toka koji se nalazi na slici 3.1 te se njime okvirno predstavlja put korisnika kroz aplikaciju i određene mogućnosti koje se pojavljuju.

Izrada aplikacije izvršavat će se u više iteracija gdje će svaka nova iteracija donositi nove značajke i funkcionalnosti. Kao što je vidljivo iz dijagrama toka aplikacije vidljivoga na slici 3.1., pokretanjem aplikacije korisniku se nudi izbornik prijave, registracije, nastavka kao gost ili promjena lozinke. Ako korisnik odabere da se želi prijaviti, dovodi ga se na unos podataka gdje se radi provjera jesu li podaci ispravni ili ne. Ako korisnik odabere da se želi registrirati, ponudi mu se da se registrira. Ako korisnik odabere promjena lozinke, vodi ga se na unos nove lozinke.



Slika 3.1. Dijagram toka aplikacije

Nakon što je odrađena prijava korisniku se nudi izbornik gdje može odabrati težinu igre, nastavak postojeće igre, prikaz rezultata i unos koda. U slučaju da korisnik odabere nastavi kao gost odmah mu započinje igra. Kada započne igru, započinje se i mjerač vremena koji će mjeriti vrijeme potrebno da korisnik dovrši igru. Nakon što je ispunio sva potrebna polja, korisnik predaje svoju zagonetku te mu se govori je li točno ili netočno riješio. Ako je točno riješena, ispiše mu se vrijeme

potrebno da riješi zagonetku te se to vrijeme spremi i prikaže mu se tipka za slanje e-maila. Ako je netočno riješena zagonetka, korisnik može nastaviti igrati.

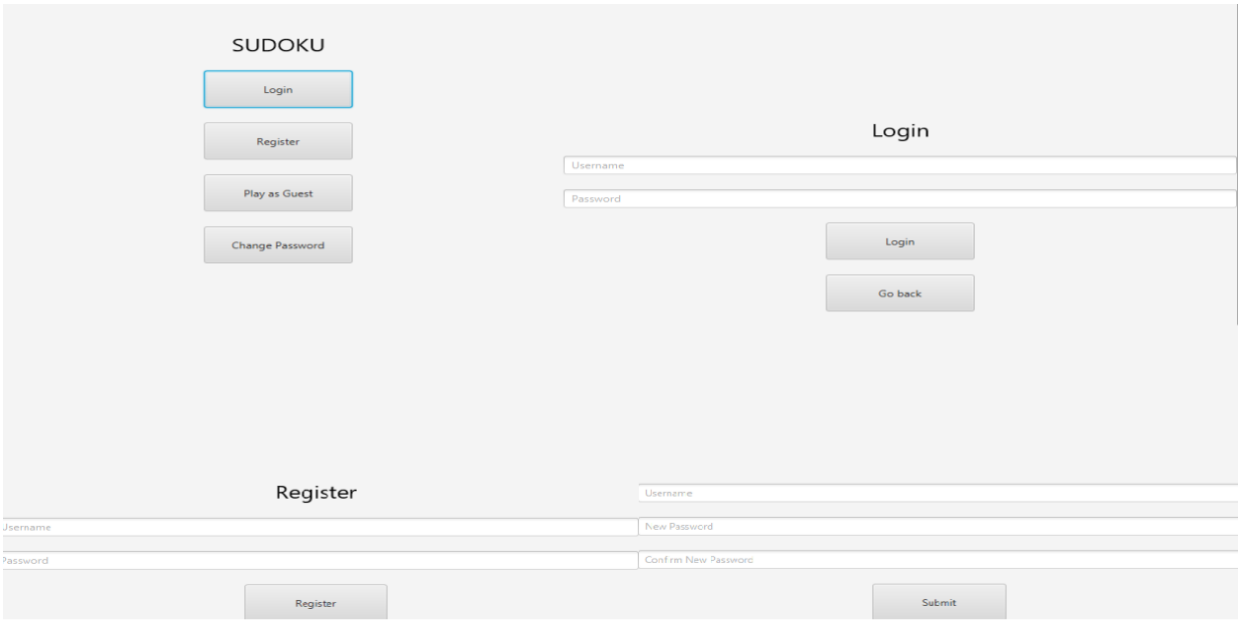
Odabirom tipke za slanje e-maila poslat će se unikatni kod te igre osobi kojoj korisnik to želi poslati.

4.1. Sučelja Sudoku desktop aplikacije

Sučelja su dizajnirana u računalnom programu Lunacy. Lunacy je besplatan alat za UI/UX i *Web* dizajn koji je podržan na Windows, macOS i Linux operacijskim sustavima. Za potrebe izrade sučelja za Sudoku desktop aplikaciju korištena je verzija 9.1 (28.06.2024.). Sučelja prikazuju korisniku opcije prijave, registracije, nastavka kao gost, započinjanja nove igre te prikaz vremena potrebnog da se riješi igra.

4.1.1. Početni zaslon, registracija i prijava

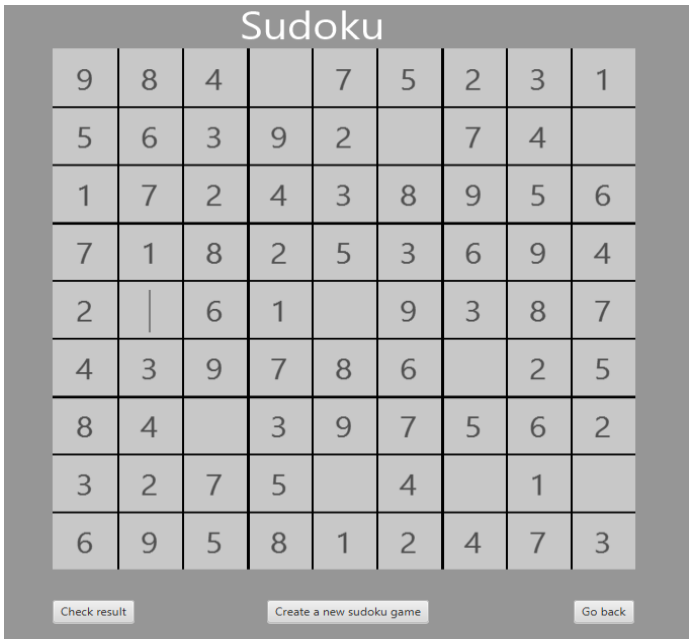
Slikom 4.1.1 predstavljen je izgled početnog zaslona, zaslona za registraciju, za prijavu i zaslona za promjenu lozinke. Na područja gdje su korisničko ime i lozinka korisnik treba unijeti svoje korisničko ime te lozinku kako bi se ili prijavio ili registrirao u aplikaciju. Na početnom zaslonu tipke *Login*, *Register*, *Play as Guest* i *Change Password* su tipke koje korisnik pritiskom na lijevu tipku miša odabire.



Slika 4.1.1. Sučelja za početni zaslon, registraciju, prijavu i promjenu lozinke

4.1.2. Sudoku mreža

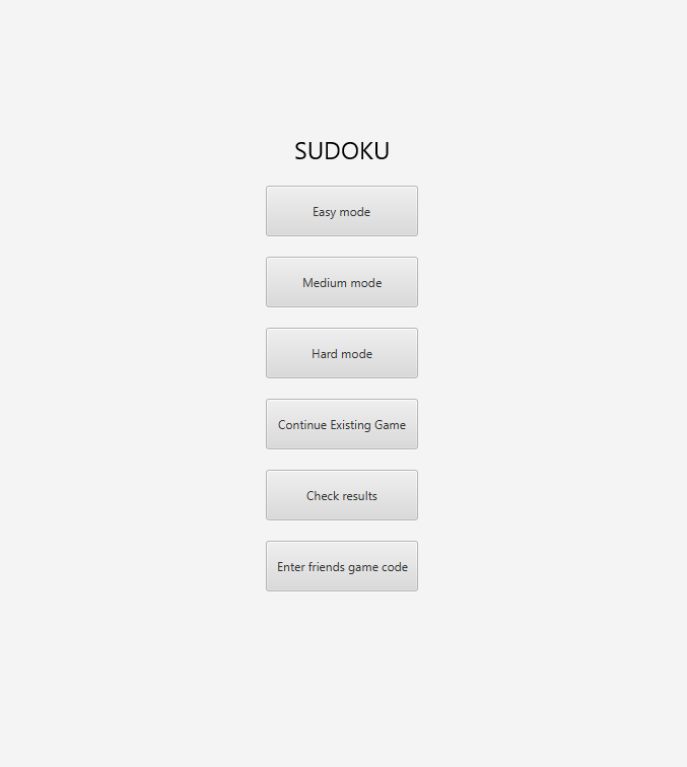
Slikom 4.1.2. prikazana je Sudoku mreža koja će biti popunjena brojevima na mjestima gdje su brojevi generirani pri generiranju zagonetke, a na mjestima gdje nema brojeva stajat će prazna mjesta. Cilj igre je da korisnik svojim poznavanjem igre Sudoku savlada zagonetku te ju riješi na točan način. Više o ovome će biti u poglavlju o generiranju Sudoku zagonetke.



Slika 4.1.2. Prikaz mreže

4.1.3. Izgled zaslona za odabir nove, nastavka postojeće igre, provjere rezultata te unosa koda

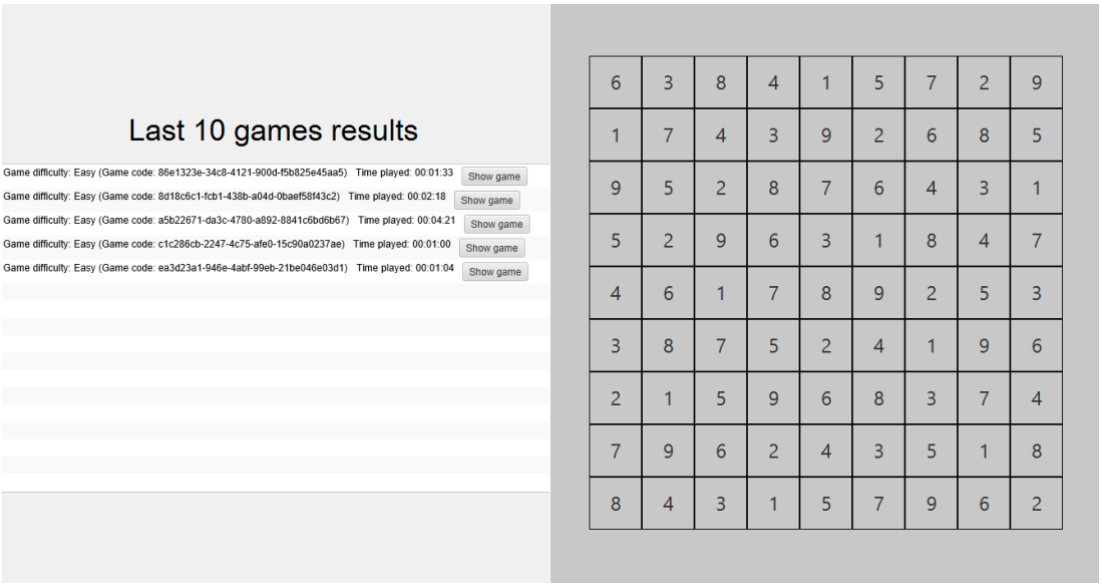
Slikom 4.1.3. prikazan je zaslon pri odabiru nove igre, nastavka postojeće igre, provjere postojećih rezultata te unos unikatnog koda za generiranje igre. Pritiskom na tipke *Easy mode, Medium mode* i *Hard mode* generirat će se Sudoku zagonetka koju će korisnik moći igrati. Pritiskom na *Continue Existing Game* Otvorit će se prozor sa zagonetkom koju je korisnik već započeo igrati te će moći nastaviti igrati. Pritiskom na *Check results* prikazat će se lista posljednjih 10 igara kojima će korisnik moći pogledati kako su riješene. Pritiskom na *Enter friends game code* otvorit će se prozor u kojemu će korisnik moći unijeti kod za generiranje igre koji je dobio na svoj mail te će ju moći rješavati.



Slika 4.1.3. Izgled zaslona za odabir nove, nastavka postojeće igre, provjere rezultata te unosa koda

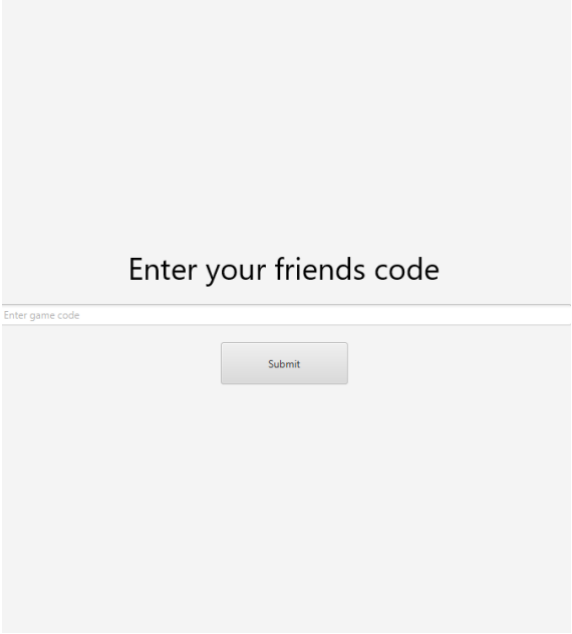
4.1.4. Izgled zaslona za provjeru rezultata te zaslona za prikaz rezultata

Slikom 4.1.4 Prikazani su izgled zaslona za provjeru rezultata te zaslona nakon što korisnik pritisne tipku *Show game*.



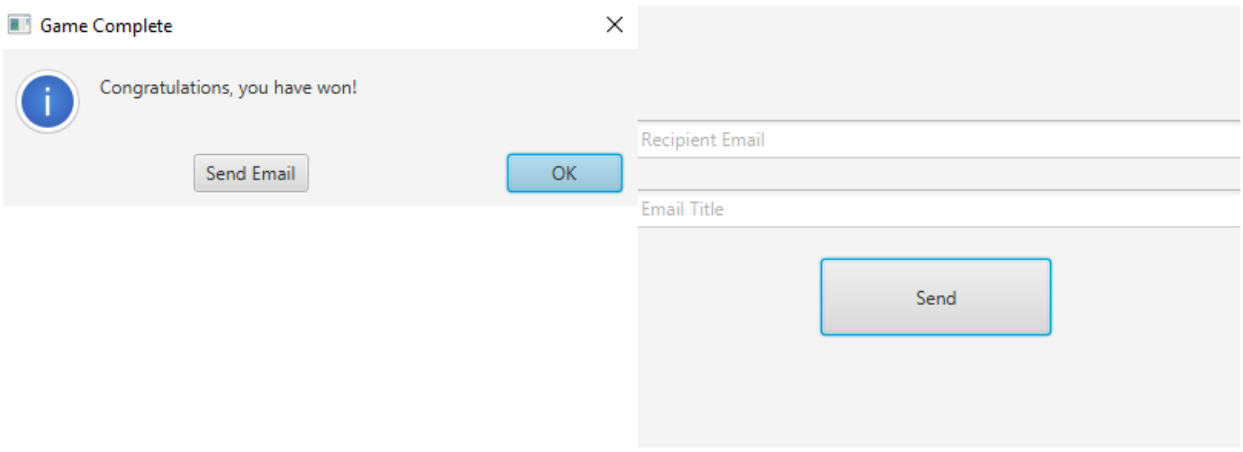
Slika 4.1.4. Izgled zaslona za provjeru rezultata te zaslona za prikaz rezultata

Slikom 4.1.5. Prikazan je izgled zaslona za unos unikatnog koda za generiranje igre kojim će korisnik moći generirati igru koja je identična igri koju je korisnik dobio na svoj mail od strane drugog korisnika.



Slika 4.1.5. Izgled zaslona za unos unikatnog koda za generiranje igre

Slikom 4.1.6. prikazani su izgled zaslona nakon uspješno riješene zagonetke te zaslona za slanje e-maila. Kada korisnik ispuni polja *Recipient Email* i *Email title* i pritisne tipku *Send* poslat će se mail koji sadrži ID, kratica za identifikacijski broj, te zagonetke.



Slika 4.1.6. Izgled zaslona nakon uspješno riješene zagonetke te zaslona za slanje e-maila

5. IZRADA APLIKACIJE

U ovom poglavlju detaljnije će biti opisano izrađeno programsko rješenje za početni zaslon, funkcionalnosti registracije i logiranja korisnika, nastavka kao gost, promjenu lozinke te podatkovni modeli i implementacija glavnog zaslona aplikacije odgovornog za rukovanjem s početkom igre, generiranjem igre iz koda, prikazom prošlih igara te nastavkom postojeće igre.

5.1. Kreiranje početnog zaslona i funkcionalnosti za login, registraciju, nastavka kao gost i promjenu lozinke

Na slici 5.1.1. vidljiva je klasa *StartingScreenController* koja je odgovorna za upravljanje početnim zaslonom aplikacije. Klasa ima jednu varijabli tipa *String* [8] koja sprema korisničko ime korisnika koji je trenutno ulogiran u aplikaciju. Sastoji se od 4 *FXML* [9] metode [10] koje, kada korisnik pritisne na gumb, odvest će korisnika na pravilni zaslon, od 2 *Public* [11] metode koje su odgovorne za postavljanje i dohvaćanje korisničkog imena (engl. *Set and Get methods*) te jedne *Private* [11] metode.

```
public class StartingScreenController {

    private String currentUsername;

    public void setCurrentUsername(String username) {
        this.currentUsername = username;
    }

    public String getCurrentUsername() {
        return currentUsername;
    }
```

Slika 5.1.1. Slika klase StartingScreenController sa varijablom currentUsername i metodama za dohvaćanje i postavljanje currentUsername varijable

Na slici 5.1.2. vidi se *handleLogin FXML* metoda koja se odrađuje pritiskom na gumb *Login*. Ona će učitati *LoginScreen.fxml* koji je odgovoran za prikazivanje zaslona za login korisnika. Zatim će se postaviti *LoginScreenController* koji je zasebna klasa odgovorna za login te će se otvoriti zaslon za *login*. Metoda *handleRegister* poziva se klikom na gumb *Register* te će se pozvati metoda *loadScreen* kojoj će biti predana putanja do *FXML* datoteke za registraciju korisnika i postavit će se *RegisterScreenController* koji je odgovoran za registraciju korisnika. Metoda

handlePlayAsGuest odradit će se klikom na gumb *Play as a guest*, postavit će varijablu *currentUsername* na vrijednost *guest* te će inicijalizirati *UserInterfaceImpl* klasu odgovornu za logiku generiranja igre koju će korisnik moći igrati. Klikom na gumb *Change Password* odrađuje se *handleChangePassword* koja će pozvati *loadScreen* metodu s putanjom do *FXML* datoteke za promjenu lozinke korisnika te će se pozvati *ChangePasswordController*. Metoda *loadScreen* pomoćna je metoda za učitavanje *FXML* datoteke i ažuriranje trenutnog zaslona.

```
@FXML
private void handleLogin(ActionEvent event) throws IOException {
    FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "LoginScreen.fxml"));
    Parent root = loader.load();
    LoginScreenController loginController = loader.getController();
    loginController.setStartingScreenController(this);
    Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
    stage.setScene(new Scene(root,  v: 680,  v1: 760));
    stage.show();
}

@FXML
private void handleRegister(ActionEvent event) throws IOException {
    loadScreen( fxmlFile: "RegisterScreen.fxml", event);
}

@FXML
private void handlePlayAsGuest(ActionEvent event) {
    try {
        currentUsername = "guest";
        Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
        UserInterfaceImpl userInterface = new UserInterfaceImpl(stage,  isGuest: true);
        SudokuBuildLogic.build(userInterface,  difficulty: 46);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@FXML
private void handleChangePassword(ActionEvent event) throws IOException {
    loadScreen( fxmlFile: "ChangePasswordScreen.fxml", event);
}

private void loadScreen(String fxmlFile, ActionEvent event) throws IOException {
    FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));
    Parent root = loader.load();
    Stage stage = (Stage) ((Button) event.getSource()).getScene().getWindow();
    stage.setScene(new Scene(root,  v: 680,  v1: 760));
    stage.show();
}
```

Slika 5.1.2. Slika handleLogin, handleRegister, handlePlayAsGuest, handleChangePassword i loadScreen metoda

Na slici 5.1.3. prikazana je klasa *LoginScreenController* u kojoj se odrađuje logika za login korisnika u aplikaciju. Pozivom metode *handleLogin* dohvatit će se vrijednost korisničkog imena i lozinke te zapisati te vrijednosti u varijable *username* i *password*. Zatim ćemo pozvati metodu *validateUser* koja se nalazi unutar *UserStorage* klase koja će potvrditi je li korisnik unio točne podatke za login. Ako je, ažurirat će se instanca *CurrentUser* singletona (engl. *Singleton*) [12] i ažurirat će se *startingScreenController* s trenutnim korisničkim imenom. Zatim poziva *loadWelcomeScreen* kako bi se zaslon prebacio na zaslon *SudokuWelcomeScreen.fxml* u kojemu

se nalaze gumbi za odabiranje težine igre, nastavak postojeće igre, prikaz rezultata i unos koda. U slučaju da je korisnik unio pogrešne podatke, prikazat će se upozorenje o pogrešci (engl. *Error alert*).

```
public class LoginScreenController {

    @FXML
    private TextField usernameField;

    @FXML
    private PasswordField passwordField;

    private UserStorage userStorage;
    private StartingScreenController startingScreenController;

    public LoginScreenController() {
        userStorage = new UserStorage();
    }

    @FXML
    private void handleLogin(ActionEvent event) {
        String username = usernameField.getText();
        String password = passwordField.getText();
        User user = userStorage.validateUser(username, password);
        if (user != null) {
            CurrentUser currentUser = CurrentUser.getInstance();
            currentUser.setUsername(user.getUsername());
            currentUser.setUserNumber(user.getUserNumber());

            startingScreenController.setCurrentUsername(username);
            loadWelcomeScreen(event, currentUser);
        } else {
            showError("Invalid username or password");
        }
    }
```

Slika 5.1.3. Klasa LoginScreenController, varijble i handleLogin metoda

Slika 5.1.4. prikazuje *RegisterScreenController* klasu te bitnije metode unutar klase. Klasa *RegisterScreenController* odgovorna je za upravljanje procesa registracije novog korisnika tako što će tražiti od korisnika unos korisničkog imena i lozinke. Odrađuje se provjera jesu li polja prazna te ako jesu, prikazat će se upozorenje o grešci koje se prikazuje s pomoću *showAlert* metode koja kao parametre prima naslov i poruku za prikazivanje pogreške. Nakon prve provjere, radi se provjera postoji li već takav korisnik u lokalnoj bazi podataka tako što će se uzeti korisničko ime koje je korisnik upisao te ga predati instanci (engl. *Instance)* [13] *UserStorage* klase zvanoj *userStorage* pozivanjem metode *userStorage.getUser* koja se koristi za dohvaćanje korisnika iz baze podataka s pomoću unesenog korisničkog imena. Ako su obje provjere prošle i nije se pokazala pogreška, generirat će se unikatan korisnički broj s pomoću metode *generateUserNumber* zvan *userNumber* te će se pokušati odraditi dodavanje korisnika u bazu

podataka. Ako je uspješno, prikazat će se informacijsko upozorenje koje će reći da je korisnik uspješno dodan.

```
@FXML
private void handleRegister() {
    String username = usernameField.getText();
    String password = passwordField.getText();

    if (username.isEmpty() || password.isEmpty()) {
        showAlert( title: "Error",  message: "Username and password cannot be empty.");
        return;
    }

    if (userStorage.getUser(username) != null) {
        showAlert( title: "Error",  message: "Username already exists.");
        return;
    }

    String userNumber = generateUserNumber();

    try {
        userStorage.addUser(new User(username, password, userNumber));
        showAlert( title: "Success",  message: "Registered user successfully. Please login to play!");
        loadLoginScreen();
    } catch (IOException e) {
        showAlert( title: "Error",  message: "Failed to register user.");
    }
}


@FXML
private void handleGoBack() throws IOException {
    loadScreen( fxmlFile: "/com/example/sudokudesktopapplication/StartingScreen.fxml");
}

private void showAlert(String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}
```

Slika 5.1.4. Klasa RegisterScreenController, handleRegister, handleGoBack i showAlert metode

Slika 5.1.5. prikazuje klasu *UserStorage* koja je odgovorna za upravljanje postojanošću korisničkih podataka, uključujući spremanje i dohvaćanje korisničkih informacija te njihovih Sudoku igara. *USER_DATA_FILE* objekt [14] tipa *File* [15] koji će reprezentirati datoteku u koju se spremaju podaci, *users* mapa (engl. *Map*) [16] koja sprema *Users* objekt, kojemu je ključ korisničko ime i *userGames* mapa koja sprema *SudokuGame* objekt kojemu je ključ korisničko ime. Metoda *addUser* prima objekt *User* te dodaje novog korisnika u mapu i poziva *saveUserData* metodu kako bi se spremilo promjene. Dohvaćanje korisnika radi se uz pomoć *getUser* metode koja će vratiti korisnika ovisno o korisničkom imenu. Validacija korisnika radi se pomoću *validateUser* metode koja će provjeriti odgovaraju li korisničko ime i lozinka podacima koji su zapisani, ako odgovaraju vratit će se *user*, ako ne odgovaraju vratit će se *null* [17]. *changePassword* metoda je koja služi za ažuriranje i spremanje nove lozinke. Metoda *saveUserGame* služi za spremanje korisnikove Sudoku igre u *userGames* mapu te pozivanje

saveUserData metode kako bi se podatci spremili. *getUserGames* metoda dohvaća korisnikovu Sudoku igru iz *userGames* mape. *saveUserData* metoda je metoda koja serializira (engl. *Serializes*) [18] *users* i *userGames* mape te ih zapisuje u datoteku koja je prikazana varijablom *USER_DATA_FILE*. Učitavanje podataka obavlja se pomoću *loadUserData* metode koja deserializira (engl. *Deserializes*) [18] podatke te popunjava *users* i *userGames* mape.

```
public class UserStorage {
    private static final File USER_DATA_FILE = new File(System.getProperty("user.home"), child: "userdata.txt");
    private Map<String, User> users;
    private Map<String, SudokuGame> userGames;

    public UserStorage() {
        users = new HashMap<>();
        userGames = new HashMap<>();
        loadUserData();
    }

    public void addUser(User user) throws IOException {
        users.put(user.getUsername(), user);
        saveUserData();
    }

    public User getUser(String username) { return users.get(username); }

    public User validateUser(String username, String password) {
        User user = users.get(username);
        if (user != null && user.getPassword().equals(password)) {
            user.setUserNumber(retrieveUserNumber(user));
            return user;
        }
        return null;
    }

    public void saveUserGame(String username, SudokuGame game) throws IOException {
        userGames.put(username, game);
        saveUserData();
    }

    public SudokuGame getUserGame(String username) { return userGames.get(username); }

    private void saveUserData() throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(USER_DATA_FILE))) {
            oos.writeObject(users);
            oos.writeObject(userGames);
        }
    }

    private void loadUserData() {
        if (USER_DATA_FILE.exists()) {
            try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(USER_DATA_FILE))) {
                users = (Map<String, User>) ois.readObject();
                userGames = (Map<String, SudokuGame>) ois.readObject();
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }

    private String retrieveUserNumber(User user) { return user.getUserNumber(); }

    public void changePassword(String username, String newPassword) throws IOException {
        User user = users.get(username);
        if (user != null) {
            user.setPassword(newPassword);
            saveUserData();
```

Slika 5.1.5. Klasa UserStorage

Na slici 5.1.6. vidljiva je klasa *ChangePasswordController* i njezina metoda *handleSubmit* koja će se pozvati kada korisnik klikne na gumb *Submit*. Metoda dohvaća uneseno korisničko ime, unesenu novu lozinku te ponovno unesenu novu lozinku i provjerava jesu li jednake te ako nisu prikazuje grešku. Ako jesu, pozvat će *userStorage.changePassword* metodu koja će promijeniti lozinku za zadano korisničko ime te će vratiti korisnika na početni zaslon pomoću *returnToStartingScreen* metode koja će učitati *StartingScreen.fxml* datoteku, no ako dođe do pogreške prikazat će se upozorenje o pogrešci.

```java
public class ChangePasswordController {

    @FXML
    private TextField usernameField;

    @FXML
    private PasswordField newPasswordField;

    @FXML
    private PasswordField confirmNewPasswordField;

    private UserStorage userStorage = new UserStorage();

    @FXML
    private void handleSubmit(ActionEvent event) {
        String username = usernameField.getText();
        String newPassword = newPasswordField.getText();
        String confirmNewPassword = confirmNewPasswordField.getText();

        if (!newPassword.equals(confirmNewPassword)) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText(null);
            alert.setContentText("\"New passwords do not match.\"");
            alert.showAndWait();
            return;
        }

        try {
            userStorage.changePassword(username, newPassword);
            showAlert( message: "Password changed successfully.", event);
        } catch (IOException e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText(null);
            alert.setContentText("Failed to change password.");
            alert.showAndWait();
            e.printStackTrace();
        }
    }
```

Slika 5.1.6. Klasa ChangePasswordController

5.2. Podatkovni modeli

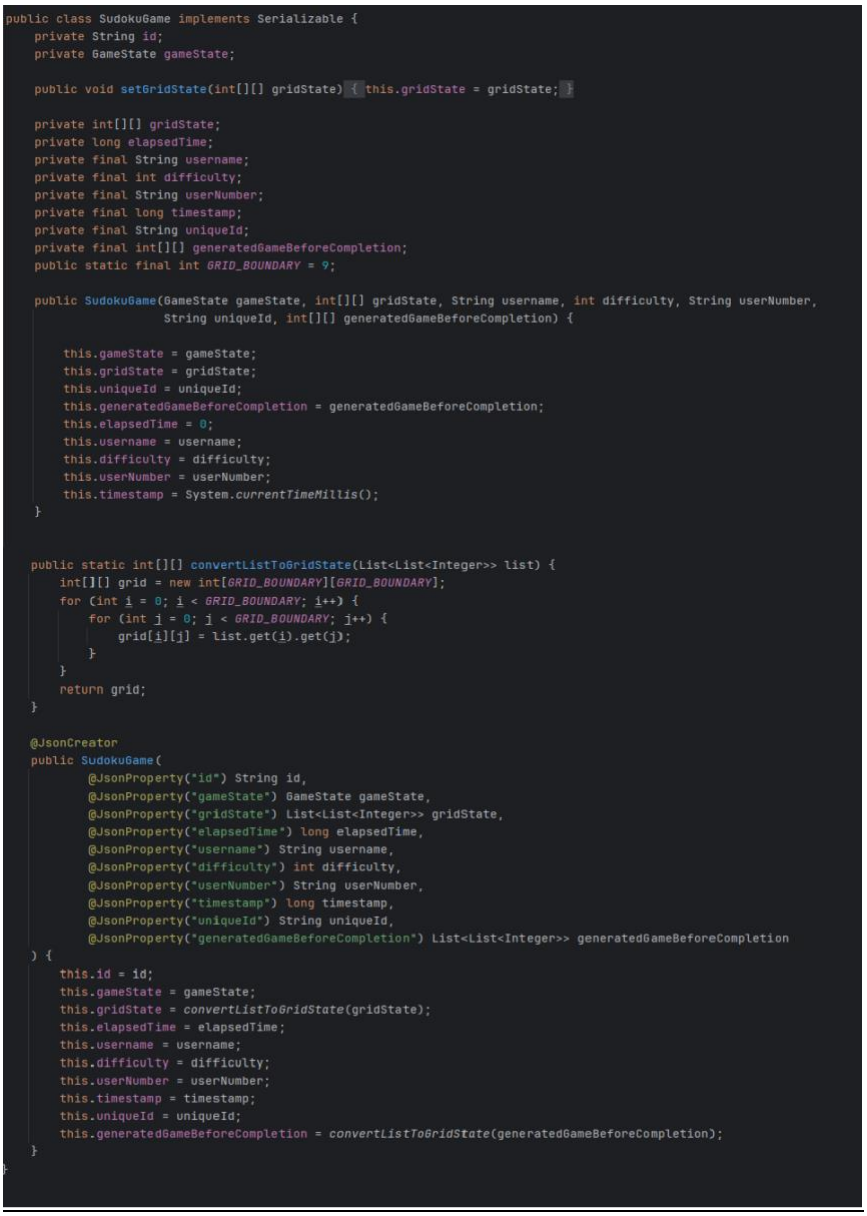
Modeli u aplikaciji koriste se kako bi se postiglo razdvajanje odgovornosti (engl. *Separation of concern*) gdje svaka komponenta radi zasebno kako bi se kod lakše razumio i unaprjeđivao. Korisno je zbog mogućnosti ponovnog korištenja modela kroz različite dijelove aplikacije.

Slika 5.2.1. prikazuje klasu *User* koja je podatkovni model koji predstavlja korisnika u aplikaciji. Enkapsulira (engl. *Encapsulates*) [19] korisnikove i omogućuje dohvaćanje i postavljanje podataka pomoću metoda za dohvaćanje i postavljanje (engl. *Getter and setter methods*) [20] te postavljanje nove lozinke pomoću *setPassword* metode.

```java
public class User implements Serializable {

    private String username;
    private String password;
    private String userNumber;

    public User(String username, String password, String userNumber) {
        this.username = username;
        this.password = password;
        this.userNumber = userNumber != null ? userNumber : UUID.randomUUID().toString();
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getUserNumber() {
        return userNumber;
    }

    public void setUserNumber(String userNumber) {
        this.userNumber = userNumber;
    }


    public void setPassword(String newPassword) {
        this.password = newPassword;
    }
}
```

Slika 5.2.1. Klasa User

Slika 5.2.2. prikazuje podatkovnu klasu *SudokuGame* koja predstavlja stanje (engl. *State*) Sudoku igre. Enkapsulira više atributa koji su vezani za igru te pruža metode za pristupanje i manipulaciju atributima.



```
public class SudokuGame implements Serializable {
    private String id;
    private GameState gameState;

    public void setGridState(int[][] gridState) { this.gridState = gridState; }

    private int[][] gridState;
    private long elapsedTime;
    private final String username;
    private final int difficulty;
    private final String userNumber;
    private final long timestamp;
    private final String uniqueId;
    private final int[][] generatedGameBeforeCompletion;
    public static final int GRID_BOUNDARY = 9;

    public SudokuGame(GameState gameState, int[][] gridState, String username, int difficulty, String userNumber,
                      String uniqueId, int[][] generatedGameBeforeCompletion) {

        this.gameState = gameState;
        this.gridState = gridState;
        this.uniqueId = uniqueId;
        this.generatedGameBeforeCompletion = generatedGameBeforeCompletion;
        this.elapsedTime = 0;
        this.username = username;
        this.difficulty = difficulty;
        this.userNumber = userNumber;
        this.timestamp = System.currentTimeMillis();
    }


    public static int[][] convertListToGridState(List<List<Integer>> list) {
        int[][] grid = new int[GRID_BOUNDARY][GRID_BOUNDARY];
        for (int i = 0; i < GRID_BOUNDARY; i++) {
            for (int j = 0; j < GRID_BOUNDARY; j++) {
                grid[i][j] = list.get(i).get(j);
            }
        }
        return grid;
    }

    @JsonCreator
    public SudokuGame(
            @JsonProperty("id") String id,
            @JsonProperty("gameState") GameState gameState,
            @JsonProperty("gridState") List<List<Integer>> gridState,
            @JsonProperty("elapsedTime") long elapsedTime,
            @JsonProperty("username") String username,
            @JsonProperty("difficulty") int difficulty,
            @JsonProperty("userNumber") String userNumber,
            @JsonProperty("timestamp") long timestamp,
            @JsonProperty("uniqueId") String uniqueId,
            @JsonProperty("generatedGameBeforeCompletion") List<List<Integer>> generatedGameBeforeCompletion
    ) {
        this.id = id;
        this.gameState = gameState;
        this.gridState = convertListToGridState(gridState);
        this.elapsedTime = elapsedTime;
        this.username = username;
        this.difficulty = difficulty;
        this.userNumber = userNumber;
        this.timestamp = timestamp;
        this.uniqueId = uniqueId;
        this.generatedGameBeforeCompletion = convertListToGridState(generatedGameBeforeCompletion);
    }
}
```

Slika 5.2.2. Klasa SudokuGame bez metoda za dohvaćanje i postavljanje

Podatci koji se nalaze u *SudokuGame* klasi su *id* koji predstavlja unikatan identifikator za igru, *gamestate* koji predstavlja trenutno stanje igre, *gridState* koji predstavlja dvodimenzionalno polje cijelih brojeva u koje se sprema trenutno stanje mreže, *elapsedTime* predstavlja utrošeno vrijeme u igru, *username* predstavlja korisničko ime, *difficulty* broj koji predstavlja koliko je odabrana igra teška, *userNumber* unikatan niz znakova za identifikaciju korisnika koji je igrao igru, *timeStamp*

broj koji predstavlja kada je igra započeta, *uniqueId* unikatan niz znakova za identifikaciju igre iz kojega se može generirati ista igra i *generatedGameBeforeCompletion* dvodimenzionalno polje cijelih brojeva koje predstavlja početno generirano stanje. Zatim se nalaze metode za dohvaćanje i postavljanje te metoda *convertListToGridState* koja je odgovorna za pretvaranje liste listi u dvodimenzionalno polje. *JsonCreator* [21] anotacija (engl. *Annotation*) [22] koristi se za inicijalizaciju polja s danim vrijednostima te pretvara *gridState* i *generatedGameBeforeCompletion* iz liste listi u dvodimenzionalno polje.

GameState enum [23] prikazan slikom 5.2.3. predstavlja jednostavno nabrajanje različitih stanja u kojemu Sudoku igra može biti. To su 3 stanja, završeno (engl. *Complete*) stanje koje govori da je igra završena, aktivno (engl. *Active*) stanje koje govori da se igra trenutno igra i novo (engl. *New*) stanje koje govori da je igra tek započeta. Klasa *Messages* je konstanta (engl. *Constant*) koja se koristi za spremanje poruka koje govore je li igra završena ili poruka greške. Enum *Rows* predstavlja nabrajanje redova u Sudoku mreži, koristi se za kategoriziranje ili identificiranje redova mreže.

```
public enum GameState {
    COMPLETE,
    ACTIVE,
    NEW
}

public class Messages {
    public static final String GAME_COMPLETE = "Congratulations, you have won!";
    public static final String ERROR = "An error has occurred.";

}

public enum Rows {
    TOP,
    MIDDLE,
    BOTTOM
}
```

Slika 5.2.3. Enumi i konstante

5.3. Implementacija glavnog zaslona aplikacije

Glavni zaslon aplikacije prikazan je *SudokuWelcomeScreen.fxml* datotekom prikazanom slikom 5.3.1. koja se koristi za definiranje korisničkog sučelja za glavni zaslon aplikacije. Koristi *JavaFX* [9] kako bi se kreiralo grafičko sučelje.

```
<VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="com.example.sudokudesktopapplication.SudokuWelcomeScreenController">
    <Text text="SUDOKU" style="-fx-font-size: 24px;"/>
    <Button text="Easy mode" onAction="#handleEasyMode" prefWidth="150" prefHeight="50"/>
    <Button text="Medium mode" onAction="#handleMediumMode" prefWidth="150" prefHeight="50"/>
    <Button text="Hard mode" onAction="#handleHardMode" prefWidth="150" prefHeight="50"/>
    <Button fx:id="continueGameButton" text="Continue Existing Game" onAction="#handleContinueGame" prefWidth="150"
            prefHeight="50"/>
    <Button text="Check results" onAction="#checkResults" prefWidth="150" prefHeight="50"/>
    <Button text="Enter friends game code" onAction="#handleEnterFriendsGameCode" prefWidth="150" prefHeight="50"/>
    <TextField fx:id="gameCodeField" promptText="Enter game code" visible="false"/>
    <VBox fx:id="listViewContainer">
        <ListView fx:id="resultsListView"/>
    </VBox>

</VBox>
```

Slika 5.3.1. SudokuWelcomeScreen.fxml datoteka

VBox je element koji se koristi kao spremnik rasporeda (engl. *Layout* container) koji uređuje ostale elemente u vertikalnom stupcu. Element *Text* će prikazati naslov „SUDOKU“ s veličinom 24 pixela [24]. Zatim su tu elementi *Button* od kojih svaki ima pojedinu funkciju. Gumb *Easy Mode* započet će novu igru s najlakšom težinom. Svaki gumb ima atribute *prefWidth* i *prefHeight* koji se odnose na širinu (engl. *Width*) te visinu (engl. *Height*) gumba.

Element *TextField* je polje za unos teksta, s atributom koji govori da će to polje biti nevidljivo (engl. *Invisible*). Gumbi mogu imati i *fx:id* atribut koji će dodijeliti identifikator određenoj komponenti. *SudokuWelcomeScreen.fxml* datoteka koristit će se u *SudokuWelcomeScreenController*-u koji je zaslužan za rukovanje s korisničkim interakcijama na glavnom zaslonu aplikacije.

Slikom 5.3.2 prikazan je *SudokuWelcomeScreenController* koji je odgovoran za upravljanje glavnog zaslona aplikacije. *currentUsername* sprema korisničko ime korisnika, *firebaseService* instanca je *FirebaseService* klase koja se koristi za implementaciju spremanja i dohvaćanja podataka iz baze podataka što će biti objašnjeno u daljnjem dijelu rada. *resultsListView* je komponenta definirana u *FXML* datoteci te se koristi za prikaz liste rezultata igre. *listViewContainer* je spremnik definiran u *FXML* datoteci koji sadrži *resultsListView*. Unutar klase nalazi se i konstruktor (engl. *Constructor*) [25] koji inicijalizira *firebaseService* tako što će

napraviti novu instancu (engl. *Instance*) [26] *FirebaseService* klase. *setCurrentUser* je metoda za postavljanje korisničkog imena kako bi se moglo koristiti u daljnjoj pohrani.

```
public class SudokuWelcomeScreenController {

    private String currentUsername;
    private FirebaseService firebaseService;

    @FXML
    private ListView<String> resultsListView;

    @FXML
    private VBox listViewContainer;

    public SudokuWelcomeScreenController() {
        this.firebaseService = new FirebaseService();
    }

    public void setCurrentUser(CurrentUser currentUser) {
        this.currentUsername = currentUser.getUsername();
    }
```

Slika 5.3.2. SudokuWelcomeScreenController

Na slici 5.3.3. prikazane su metode koje se koriste za kreiranje nove Sudoku igre ovisno o korisnikovom izboru. Metoda *handleEasyMode* koristi se kada korisnik odabere gumb *Easy Mode* te poziva metodu *startGameWithDifficulty* koja kao parametre prima *ActionEvent* događaj [27] koji se odnosi na događaj koji je pokrenuo metodu, u ovom slučaju pritisak na gumb *Easy Mode* i *String difficulty* koji se odnosi na težinu igre. Metoda će dohvatiti instancu *currentUser* trenutnog korisnika, koristiti *switch statement* [28] kako bi se odredila popunjenost polja u trenutku započinjanja igre. Laka (engl. *Easy*) težina imat će 35 popunjenih polja, srednja (engl. *Medium*) težina imat će 30, a teška (engl. *Hard*) težina imat će 25 popunjenih polja. Zatim će se pokušati započeti igra tako što će se napraviti nova instanca *UserInterfaceImpl* klase te će se pozvati *SudokuBuildLogic.build* metoda koja će generirati igru. Detaljnije objašnjenje *UserInterfaceImpl* te *SudokuBuildLogic* klasa navedeno je u daljnjem tekstu. Logika je ista za *handleMediumMode* i *handleHardMode* metode.

```
@FXML
private void handleEasyMode(ActionEvent event) {
    startGameWithDifficulty(event, difficulty: "easy");
}

@FXML
private void handleMediumMode(ActionEvent event) {
    startGameWithDifficulty(event, difficulty: "medium");
}

@FXML
private void handleHardMode(ActionEvent event) {
    startGameWithDifficulty(event, difficulty: "hard");
}

private void startGameWithDifficulty(ActionEvent event, String difficulty) {
    CurrentUser currentUser = CurrentUser.getInstance();
    int difficultyLevel;
    switch (difficulty) {
        case "easy":
            difficultyLevel = 46;
            break;
        case "medium":
            difficultyLevel = 51;
            break;
        case "hard":
            difficultyLevel = 56;
            break;
        default:
            throw new IllegalArgumentException("Unknown difficulty: " + difficulty);
    }

    try {
        Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
        UserInterfaceImpl userInterface = new UserInterfaceImpl(stage, isGuest: false);
        SudokuBuildLogic.build(userInterface, difficultyLevel, currentUsername, currentUser.getUserNumber());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Slika 5.3.3. SudokuWelcomeScreenController metode za kreiranje nove igre

Metoda *handleContinueGame* prikazana slikom 5.3.4. odgovorna je za nastavak lokalno spremljene Sudoku igre za trenutno ulogiranog korisnika. Kreirat će instancu *UserStorage* klase kako bi se moglo pristupati spremljenim podatcima, dohvatit će trenutnog korisnika koristeći *getInstance* metodu iz *CurrentUser* singletona te će dohvatiti igru za trenutnog korisnika pozivajući *userStorage.getUserGame(currentUser.getUsername())* te će provjeriti postoji li spremljena igra i ako postoji, prikazat će ju korisniku tako što će napraviti novu instancu *UserInterfaceImpl* klase i pozvati *SudokuBuildLogic.build* metodu s napravljenim sučeljem (engl. *Interface*) i pohranjenom igrom, no ako ne postoji prikazat će se upozorenje o pogrešci. *checkResults* metoda poziva se kada korisnik pritisne na gumb *Check results* te je odgovorna za provjeru i prikazivanje posljednje odigranih 10 igara za trenutno ulogiranog korisnika. Metoda će dohvatiti trenutnog korisnika pomoću *CurrentUser.getInstance()* metode, provjeriti da korisnik ima postavljen *userNumber* te će onda odraditi dohvaćanje igara pomoću *firebaseService.getLast10Games* metode koja je implementirana u *FirebaseService* klasi. *onCallBack* metoda je dio sučelja (engl. *Interface*) ili apstraktne klase [29]. Koristi se unutar klase *FirebaseService* unutar *FirebaseCallback* sučelja kako bi se rukovalo asinkronim operacijama

(engl. *Asynchronous operations*) [30] za dohvaćanje podataka iz *Firebase*-a. Metoda je dizajnirana da se pozove kada se asinkrona operacija završi. Koristi se tako da rukuje s listom 10 zadnjih igara spremljenih za trenutnog korisnika koje su dohvaćene iz baze podataka *Firebase*.

```
@FXML
private void handleContinueGame(ActionEvent event) {
    UserStorage userStorage = new UserStorage();
    CurrentUser currentUser = CurrentUser.getInstance();
    SudokuGame savedGame = userStorage.getUserGame(currentUser.getUsername());
    if (savedGame != null) {
        try {
            Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
            UserInterfaceImpl userInterface = new UserInterfaceImpl(stage, isGuest: false);
            SudokuBuildLogic.build(userInterface, savedGame);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else {
        showError("No saved game found for user: " + currentUser.getUsername());
    }
}

@FXML
private void checkResults(ActionEvent event) {
    CurrentUser currentUser = CurrentUser.getInstance();
    if (currentUser.getUserNumber() == null || currentUser.getUserNumber().isEmpty()) {
        showError("User number is not set. Please log in again.");
        return;
    }

    firebaseService.getLast10Games(currentUser.getUserNumber(), new FirebaseService.FirebaseCallback() {
        @Override
        public void onCallback(List<SudokuGame> last10Games) {
            try {
                FXMLLoader loader = new FXMLLoader(getClass().
                        getResource( name: "/com/example/sudokudesktopapplication/ResultsWindow.fxml"));
                Parent root = loader.load();
                ResultsWindowController controller = loader.getController();
                controller.setResults(last10Games);

                Stage stage = new Stage();
                stage.setTitle("Results");
                stage.setScene(new Scene(root, v: 680, v1: 760));
                stage.show();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
}
```

Slika 5.3.4. SudokuWelcomeScreenController metode za nastavak postojeće igre te dohvaćanje rezultata igara

Slika 5.3.5. prikazuje metode *handleGameSelection* i *showGameDetails* koje su odgovorne za rukovanje korisničkim interakcijama za odabir i prikaz detalja odigrane Sudoku igre. *handleGameSelection* metoda poziva se kada korisnik odabere jednu od prikazanih igara pritiskom na *Show game* gumb. Metoda će provjeriti ako je nešto odabrano te će za to dohvatiti igru pomoću *firebaseService.getGameById* koristeći *id* od igre i trenutnog korisnika *userNumber* te će se koristiti *onCallback* kako bi se rukovalo s dohvaćenim rezultatima. Ako dohvaćeni podatci postoje, poziva se *showGameDetails* metoda. *showGameDetails* metoda prikazuje detalje odigrane Sudoku igre tako što će se kreirati prozor za prikaz te će se postaviti naslov pomoću

alert.setTitle, zaglavlje *setHeaderText* te kontekst *setContextText* kojemu će se predati detalji igre koji su dohvaćeni.

```
@FXML
private void handleGameSelection() {
    String selectedItem = resultsListView.getSelectionModel().getSelectedItem();
    CurrentUser currentUser = CurrentUser.getInstance();

    if (selectedItem != null) {
        String gameId = selectedItem.substring(selectedItem.indexOf("(ID: ") + 5, selectedItem.length() - 1);
        firebaseService.getGameById(currentUser.getUserNumber(), gameId, new FirebaseService.FirebaseCallback() {
            @Override
            public void onCallback(List<SudokuGame> games) {
                if (!games.isEmpty()) {
                    SudokuGame game = games.get(0);
                    showGameDetails(game);
                }
            }
        });
    }
}

private void showGameDetails(SudokuGame game) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Game Details");
    alert.setHeaderText("Game ID: " + game.getId());
    alert.setContentText("Difficulty: " + game.getDifficulty() + "\n" +
            "Elapsed Time: " + game.getElapsedTime() + " seconds\n" +
            "Timestamp: " + new java.text.SimpleDateFormat( pattern: "HH:mm:ss")
            .format(new java.util.Date(game.getTimestamp())));
    alert.showAndWait();
}
```

Slika 5.3.5. SudokuWelcomeScreenController metode za odabir igre i prikaz detalja odabrane igre

Slikom 5.3.6. prikazane su metode *handleEnterFriendsGameCode* koja je odgovorna za otvaranje novog prozora gdje korisnik može unijeti kod koji je dobio na mail kako bi se učitala jedinstvena igra koju je netko već generirao i odigrao i metoda *loadGameFromCode* koja će učitati Sudoku igru generiranu za kod koji korisnik unosi. Metoda *handleEnterFriendsGameCode* učitava *FXML* datoteku, dohvaća kontroler (engl. *Controller*) *fxmlLoader.getController()*, postavlja metodu koja će se pozvati kada se unese kod loadGameFromCode te kreira i prikazuje novi prozor dimenzija 680x760. *loadFromCode* metoda odgovorna je za učitavanje Sudoku igre iz koda kojega korisnik unosi. Dohvaća trenutni prozor, kreira instancu *UserInterfaceImpl* s trenutnim prozorom i zastavicom (engl. *Boolean flag*) te poziva *buildFromGeneratedGameBeforeCompletion* metodu iz *SudokuBuildLogic* klase kako bi se kreirala igra.

```java
@FXML
private void handleEnterFriendsGameCode() {
    try {
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().
                getResource( name: "/com/example/sudokudesktopapplication/EnterGameCodeWindow.fxml"));
        Parent root = fxmlLoader.load();
        EnterGameCodeWindowController controller = fxmlLoader.getController();
        controller.setOnGameCodeEntered(this::loadGameFromCode);
        Stage stage = new Stage();
        stage.setTitle("Enter Game Code");
        stage.setScene(new Scene(root, v: 680, v1: 760));
        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void loadGameFromCode(SudokuGame game) {
    try {
        Stage stage = (Stage) resultsListView.getScene().getWindow();
        UserInterfaceImpl userInterface = new UserInterfaceImpl(stage, isGuest: false);
        SudokuBuildLogic.buildFromGeneratedGameBeforeCompletion(userInterface, game);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Slika 5.3.6. SudokuWelcomeScreenController metode za unos koda i kreiranje igre iz koda

6. DOHVAĆANJE I SPREMANJE U BAZU PODATAKA

Dohvaćanje i spremanje podataka u bazu podataka ključno je za normalno funkcioniranje aplikacije. U poglavljima 6.1. i 6.2. detaljnije su opisani pohrana i dohvaćanje podataka.

6.1. Pohrana podataka

Pohrana podataka odvija se unutar klase *FirebaseService* unutar koje su kreirane metode za interakciju s *Firebase*-om, specifično za spremanje i dohvaćanje podataka o Sudoku igri. Unutar konstruktora odrađuje se inicijalizacija Firebase aplikacije i postavljanje reference na Sudoku čvor (engl. *Node*) [32] u bazi podataka. Poziva se *initializeFirebase* metoda iz *FirebaseInitializer* klase ako bi se postavili potrebni podaci te se dohvaća instanca baze podataka koristeći *Firebase*-ovu klasu *FirebaseDatabase* te se poziva metoda *getInstance* te se zatim postavlja referenca baze podataka pomoću *database.getReference* kako bi se pokazivalo na čvor Sudoku.

Slikom 6.1.1. prikazana je metoda SaveGame kojoj je glavna svrha spremiti *SudokuGame* objekt u *Firebase Realtime Database*. Metoda radi na način da prvo pretvori *SudokuGame* objekt u mapu koristeći *gameToMap* metodu prikazanu na slici 6.1.2. koja je korisna za serijalizaciju podataka o igri u *JSON* format kako bi se spremila u bazu podataka. Inicijalizirat će *HashMap* [33] kako bi se spremali podatci, zatim dodaje stanje igre *gameState* u mapu pretvarajući ga u String tip podatka pomoću *toString()* [34] metode, zatim dodaje stanje mreže *gridState* u mapu pretvarajući ga u listu pomoću *getGridStateAsList()* metode, zatim dodaje utrošeno vrijeme *elapsedTime* u mapu pomoću metode *getElapsedTime()*, dodaje korisničko ime u mapu dohvaćeno metodom *getUsername()*, dodaje težinu *difficulty* pomoću *getDifficulty()* metode, isto tako i za jedinstveni korisnički broj userNumber pomoću metode *getUserNumber()*, *timestamp* pomoću *getTimestamp()* metode, jedinstveni identifikator igre *id* pomoću *getUniqueId()* te dodaje generiranu igru koja je spremljena prije nego je završena generatedGameBeforeCompletion pomoću *getGeneratedGameBeforeCompletion()* metode i zatim vraća natrag mapu. Nakon pretvaranja objekta *SudokuGame* u mapu, dohvatit će se jedinstveni identifikator igre *id* te će se kreirati *URL* za specifičnu igru. Nakon toga otvorit će se *HTTP* [35] veza za kreirani *URL* pomoću *(HttpURLConnection) url.openConnection()*, zatim će se postaviti metoda za slanje zahtjeva (engl. *Request Method*) [36] metodom *setRequestMethod()* postavljena na *PUT* metodu, koja se koristi za ažuriranje ili kreiranje podatka. Nakon toga postavljaju se postavke zahtjeva metodom *setRequestProperty()* kao što su tip sadržaja (engl. *Content type*) i vrijednost podatka, u ovom

slučaju bit će *JSON*. Nadalje, vezi se omogućuje metodom setDoOutput() da se podatci pošalju. Podatci se pomoću metode *new ObjectMapper().writeValueAsString()* [36] pretvaraju u *JSON*. Nakon toga, podatci će se, pomoću *connection.getOutputStream()* metode zapisati u izlazni podatkovni tok (engl. *Output stream*) veze te će se dohvatiti kod odgovora (engl. *Response code*) [38] pomoću getResponseCode() metode i radit će se provjera ako odgovara *HTTP_OK* statusu koji govori da je *HTTP* metoda uspješno odrađena. Ako je vraćen bilo koji drugi kod prikazat će se upozorenje o pogrešci.

```
public void saveGame(SudokuGame game) {
    Map<String, Object> gameData = gameToMap(game);
    String uniqueId = game.getUniqueId();

    try {
        URL url = new URL( spec: "https://sudoku-39493-default-rtdb.europe-west1.firebasedatabase.app/games/"
                + uniqueId + ".json");
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("PUT");
        connection.setRequestProperty("Content-Type", "application/json; utf-8");
        connection.setDoOutput(true);

        String jsonInputString = new ObjectMapper().writeValueAsString(gameData);

        try (OutputStream os = connection.getOutputStream()) {
            byte[] input = jsonInputString.getBytes( charsetName: "utf-8");
            os.write(input,  off: 0, input.length);
        }

        int responseCode = connection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            System.out.println("Game saved with ID: " + uniqueId);
        } else {
            System.err.println("Failed to save game data.");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Slika 6.1.1. saveGame metoda u klasi FirebaseService

```
private Map<String, Object> gameToMap(SudokuGame game) {
    Map<String, Object> map = new HashMap<>();
    map.put("gameState", game.getGameState().toString());
    map.put("gridState", game.getGridStateAsList());
    map.put("elapsedTime", game.getElapsedTime());
    map.put("username", game.getUsername());
    map.put("difficulty", game.getDifficulty());
    map.put("userNumber", game.getUserNumber());
    map.put("timestamp", game.getTimestamp());
    map.put("uniqueId", game.getUniqueId());
    map.put("generatedGameBeforeCompletion", game.getGeneratedGameBeforeCompletion());
    return map;
}
```

Slika 6.1.2. GameToMap metoda

6.2. Dohvaćanje podataka

Dohvaćanje podataka prikazano je slikom 6.2.1. koja prikazuje metodu *getLast10Games*.

Metoda će najprije provjeriti postoji li jedinstveni korisnički broj te će konstruirati *URL* pomoću kojega će se slati upit na bazu podataka kako bi se dohvatilo posljednjih 10 igara za trenutno ulogiranog korisnika. Zatim će se kao i pri pohrani podataka otvarati veza prema bazi, postavljat će se metoda za slanje zahtjeva, u ovom slučaju to će biti *GET* metoda koja se koristi za dohvaćanje podataka, zatim će se dohvatiti kod odgovora te će se provjeriti odgovara li kod *HTTP_OK* kodu te će se, ako je kod odgovara, čitati odgovor. Odgovor se čita tako što se kreira *BufferReader* [39] koji će čitati ulazni tok podataka veze te će se taj tok čitati liniju po liniju i dodavati u *StringBuilder* [40] zatim će se *JSON* odgovor raščlaniti (parse) pomoću *.readValue* metode iz *ObjectMapper* klase u mapu podataka odgovora. Nakon toga radi se provjera ako korisnik nema podataka vezanih za igre te ako nema, prikazuje se upozorenje o pogrešci. Ako su podaci postojani, pretvaraju se tako što se inicijalizira lista objekata *SudokuGame* te se prolazi kroz sve podatke u mapi podataka odgovora, pretvara svaki podatak u *SudokuGame* objekt i dodaje ga se u listu.

```java
public void getLast10Games(String userNumber, final FirebaseCallback callback) {
    if (userNumber == null || userNumber.isEmpty()) {
        throw new IllegalArgumentException("User number cannot be null or empty");
    }
    try {
        String urlString = "https://sudoku-39493-default-rtdb.europe-west1.firebasedatabase.app/games.json?orderBy="
                + "\"userNumber\"&equalTo=\"" + userNumber + "\"&limitToLast=10";
        URL url = new URL(urlString);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        int responseCode = connection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            ObjectMapper objectMapper = new ObjectMapper();
            List<SudokuGame> games = new ArrayList<>();
            Map<String, Object> responseData = objectMapper.readValue(response.toString(), Map.class);

            if (responseData == null) {
                showAlert("No games found.");
                return;
            }
            for (Map.Entry<String, Object> entry : responseData.entrySet()) {
                SudokuGame game = objectMapper.convertValue(entry.getValue(), SudokuGame.class);
                games.add(game);
            }
            callback.onCallback(games);
        } else {
            System.err.println("Failed to retrieve games.");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Slika 6.2.1. getLast10Games metoda za dohvaćanje posljednjih 10 odigranih igara

7. GENERIRANJE IGRE SUDOKU

U ovom poglavlju cilj je objasniti izgradnju i generiranje Sudoku igre te provjeru rješivosti pojedine igre.

7.1. Logika za izgradnju Sudoku igre

Klasa SudokuBuildLogic, prikazana slikom 7.1.1. odgovorna je za postavljanje Sudoku igre, inicijalizira stanje igre, ažurira bazu podataka te konfigurira korisničko sučelje. Sastoji se od 4 metode, 3 metode istoga imena *build()* koje se razlikuju po parametrima koje primaju i logici koju obavljaju te metode *buildFromGeneratedGameBeforeCompletion()*.

```
public class SudokuBuildLogic {

    public static void build(UserInterfaceImpl userInterface, int difficulty, String username, String userNumber)
            throws IOException {
        Storage storage = new LocalStorage(username);
        FirebaseService firebaseService = new FirebaseService();
        User user = new User(username, password: "defaultPassword", userNumber);
        SudokuGame initialState = GameLogic.getNewGame(difficulty, user);
        storage.updateGameData(initialState);

        UserInterfaceContract.EventListener uiLogic = new ControlLogic(storage, userInterface, difficulty,
                username, firebaseService, user);

        userInterface.setListener(uiLogic);
        userInterface.updateBoard(initialState);
    }

    public static void build(UserInterfaceImpl userInterface, int difficulty) throws IOException {
        build(userInterface, difficulty, username: "guest", userNumber: "guestUserNumber");
    }
```

Slika 7.1.1. Build metode unutar klase SudokuBuildLogic

Metoda *build* koja prima instancu klase *UserInterfaceImpl* nazvanu *userInterface*, težinu (engl. *Difficulty*), korisničko ime (engl. *Username*) te jedinstveni korisnički broj (engl. *UserNumber*) odgovorna je za inicijalizaciju nove Sudoku igre s definiranom težinom, korisničkim imenom te jedinstvenim korisničkim brojem na način da inicijalizira instancu *LocalStorage* klase za predano korisničko ime, inicijalizira instancu *FirebaseService* klase te kreira objekt *User* s podatcima predanim metodi. Zatim će se kreirati nova igra tako što će se pozvati *GameLogic.getNewGame* kojemu će se predati težina i objekt korisnik te će se ažurirati lokalna pohrana s inicijalnom igrom. Nadalje kreira se nova logika za kontroliranje korisnikovih unosa inicijaliziranjem instance *ControlLogic* klase te se postavlja slušatelj događaja (engl. *Event Listener*) [41] i zatim se ažurira korisničko sučelje kako bi se prikazalo početno stanje igre.

Metoda *build* koja prima instancu klase *UserInterfaceImpl* nazvanu *userInterface* i težinu pozvat će prvu metodu *build* s podatcima za korisnika koji se nije logirao niti registrirao.

7.2. Logika za generiranje Sudoku igre

GameLogic klasa prikazana slikom 7.2.1. sadrži glavnu logiku generiranja i validiranja Sudoku igre. U njoj se nalaze metode za kreiranje nove igre, provjeru stanja igre te validaciju Sudoku mreže. U daljnjem tekstu opisane su metode, njihova svrha i implementacija.

```
public class GameLogic {

    public static SudokuGame getNewGame(int difficulty, User user) {
        int[][] newGameGrid = GameGenerator.getNewGameGrid(difficulty);
        String uniqueId = UUID.randomUUID().toString();
        int[][] generatedGameBeforeCompletion = SudokuUtilities.copyToNewArray(newGameGrid);
        return new SudokuGame(GameState.NEW, newGameGrid, user.getUsername(), difficulty,
                user.getUserNumber(), uniqueId, generatedGameBeforeCompletion);
    }

    public static GameState checkForCompletion(int[][] grid) {
        if (sudokuIsInvalid(grid)) return GameState.ACTIVE;
        if (tilesAreNotFilled(grid)) return GameState.ACTIVE;
        return GameState.COMPLETE;
    }

    static boolean sudokuIsInvalid(int[][] grid) {
        if (rowsAreInvalid(grid)) return true;
        if (columnsAreInvalid(grid)) return true;
        if (squaresAreInvalid(grid)) return true;
        else return false;

    }
```

Slika 7.2.1. Metode GameLogic klase

Metoda *getNewGame* ima svrhu generiranja nove sudoku igre s njoj predanim podatcima o težini i objektu s korisnikovim podatcima. Poziva *GameGenerator.getNewGameGrid* metodu iz klase *GameGenerator* koja je prikazana na slici 7.2.2., detaljnije objašnjenoj u daljnjem tekstu, kako bi se generirala Sudoku mreža. Kada se mreža generirala, generira se jedinstveni identifikator igre *uniqueId* pomoću *randomUUID()* [42] metode te kopira generiranu mrežu u varijablu *generatedGameBeforeCompletion* koristeći pomoćnu metodu *copyToNewArray* iz klase *SudokuUtilities* koja služi za kopiranje jednog dvodimenzionalnog polja u drugo dvodimenzionalno polje. Metoda će vratiti novi objekt *SudokuGame* klase inicijaliziran s podatcima o stanju igre, mrežom, korisničkim imenom, težinom, jedinstvenim korisničkim brojem, jedinstvenim identifikatorom igre i generiranom igrom prije završetka. Metoda *checkForCompletion* provjerava je li igra završena tako što će pozvati *sudokiIsInvalid* metodu s predanom mrežom te ako je to istinito (engl. *True*) onda će postaviti stanje igre u aktivno *ACTIVE*, isto tako i za metodu *tilesAreNotFilled*, no ako ovi uvjeti nisu ispunjeni, onda je igra završena te se vraća završeno stanje *COMPLETE*.

Metoda *SudokuIsInvalid* ima svrhu provjere je li Sudoku mreža netočna. Poziva 3 metode koje vraćaju istinu ili laž (engl. *False*) ovisno te će za svaku od njih vratiti istinu što će rezultirati time

da je mreža netočno popunjena. Metoda *rowsAreInvalid* ima za svrhu provjeriti ako je bilo koji red Sudoku mreže netočno popunjen. Prolazit će kroz svaki red te će sakupljati vrijednosti u listu. Zatim poziva metodu *collectionHasRepeats* koja provjerava postoje li dupliciranih vrijednosti u retku. Ako red ima dupliciranih vrijednosti, vratit će true, inače vraća false. Sljedeća metoda za provjeru je *columnsAreInvalid* te njena svrha je provjeriti ima li neispravnih stupaca. Prolazi kroz svaki stupac i sakuplja vrijednosti u listu te poziva *collectionHasRepeats* te je ostatak logike isti kao i kod *rowsAreInvalid* metode. Metoda *squaresAreInvalid* provjerava 3 × 3 kvadrate jesu li popunjeni kako bi trebali biti. Poziva metode *rowOfSquaresIsInvalid* metodu s gornjim (engl. *Top*), srednjim (engl. *Middle*) i donjim (engl. *Bottom*) redovima kako bi se provjerili redovi tog 3 × 3 kvadrata. Vraća true ako je bilo koji 3 × 3 kvadrat netočno riješen, inače vraća false. Metoda *rowOfSquaresIsInvalid* provjerava je li specifičan redak 3 × 3 kvadrata netočno riješen. Ovisno o predanom retku poziva *squareIsInvalid* metodu za svaki 3 × 3 kvadrat u tom retku. Metoda *squareIsInvalid* provjerava je li određeni 3 × 3 kvadrat u Sudoku mreži netočno riješen. Sakuplja vrijednosti kvadrata počevši od x i y indeksa u listu i poziva *collectionHasRepeats* kako bi provjerila postoje li duplikati u kvadratu. Metoda *collectionHasRepeats* služi za provjeru ima li lista cijelih brojeva duplikate. Prolazi kroz brojeve od 1 do 9 te koristi *Collections.frequency* [43] kako bi provjerila ako se broj pojavljuje više od 1 puta. Metoda *tilesAreNotFilled* provjerava ima li praznih polja u Sudoku mreži.

```java
public class GameGenerator {
    public static int[][] getNewGameGrid(int difficulty) {
        return unsolveGame(getSolvedGame(), difficulty);
    }

    private static int[][] unsolveGame(int[][] solvedGame, int difficulty) {
        Random random = new Random(System.currentTimeMillis());

        boolean solvable = false;
        int[][] solvableArray = new int[GRID_BOUNDARY][GRID_BOUNDARY];

        while (!solvable) {
            SudokuUtilities.copySudokuArrayValues(solvedGame, solvableArray);

            int index = 0;

            while (index < difficulty) {
                int xCoordinate = random.nextInt(GRID_BOUNDARY);
                int yCoordinate = random.nextInt(GRID_BOUNDARY);

                if (solvableArray[xCoordinate][yCoordinate] != 0) {
                    solvableArray[xCoordinate][yCoordinate] = 0;
                    index++;
                }
            }

            int[][] toBeSolved = new int[GRID_BOUNDARY][GRID_BOUNDARY];
            SudokuUtilities.copySudokuArrayValues(solvableArray, toBeSolved);

            solvable = SudokuSolver.puzzleIsSolvable(toBeSolved, difficulty);
        }

        return solvableArray;
    }
```

Slika 7.2.2. Metode getNewGameGrid i unsolveGame

Metoda *getNewGameGrid* ima svrhu generiranja nove Sudoku igre ovisno o predanoj težini. Poziva metodu *getSolvedGame* prikazanoj na slici 7.2.3 kako bi se generirala potpuno riješena Sudoku igra, zatim poziva *unsolveGame* metodu za uklanjanje brojeva iz mreže koje će korisnik unositi. *unsolveGame* metoda inicijalizirat će objekt tipa *Random* [44] za generiranje slučajnih brojeva, zatim kreira kopiju riješene mreže, na slučajan odabir uklanja brojeve iz mreže dok se ne dođe do definirane težine, zatim poziva *SudokuSolver.puzzleIsSolvable* kako bi se potvrdilo da je mreža rješiva te vraća modificiranu mrežu.

Slikom 7.2.3 prikazana je metoda *getSolvedGame* koja generira potpuno riješenu Sudoku mrežu. Isto kao *unsolveGame* inicijalizirat će objekt tipa *Random* za generiranje slučajnih brojeva, kreirat će praznu 9 × 9 mrežu te će pokušati postaviti brojeve od 1 do 9 u mrežu bez kršenja pravila Sudoku igre. Koristi *sudokuIsInvalid* metodu iz klase *GameLogic* kako bi se provjerila mreža nakon svakog postavljanja. U slučaju da postavljanje brojeva previše puta bude krivo, isponova pokušava generirati mrežu. Vraća potpunu riješenu mrežu za Sudoku igru.

```java
private static int[][] getSolvedGame() {
    Random random = new Random(System.currentTimeMillis());
    int[][] newGrid = new int[GRID_BOUNDARY][GRID_BOUNDARY];

    for (int value = 1; value <= GRID_BOUNDARY; value++) {
        int allocations = 0;
        int interrupt = 0;

        List<Coordinates> allocTracker = new ArrayList<>();

        int attempts = 0;

        while (allocations < GRID_BOUNDARY) {
            if (interrupt > 200) {
                allocTracker.forEach(coord -> {
                    newGrid[coord.getX()][coord.getY()] = 0;
                });

                interrupt = 0;
                allocations = 0;
                allocTracker.clear();
                attempts++;

                if (attempts > 500) {
                    clearArray(newGrid);
                    attempts = 0;
                    value = 1;
                }
            }
            int xCoordinate = random.nextInt(GRID_BOUNDARY);
            int yCoordinate = random.nextInt(GRID_BOUNDARY);

            if (newGrid[xCoordinate][yCoordinate] == 0) {
                newGrid[xCoordinate][yCoordinate] = value;

                if (GameLogic.sudokuIsInvalid(newGrid)) {
                    newGrid[xCoordinate][yCoordinate] = 0;
                    interrupt++;
                } else {
                    allocTracker.add(new Coordinates(xCoordinate, yCoordinate));
                    allocations++;
                }
            }
        }
    }
    return newGrid;
}
```

Slika 7.2.3. Metode getNewGameGrid i unsolveGame

7.3. Provjera rješivosti Sudoku igre

Klasa *SudokuSolver* prikazana na slici 7.3.1. odgovorna je za provjeru je li generirana Sudoku igra rješiva. Koristi algoritam za popunjavanje praznih polja igre te provjeru validnosti svakog popunjenog polja. Korišten je algoritam s uzorkom na algoritam obrađen na Sveučilištu Cornell [45] [46].

```
public class SudokuSolver {
    public static boolean puzzleIsSolvable(int[][] puzzle, int difficulty) {
        Coordinates[] emptyCells = typeWriterEnumerate(puzzle, difficulty);

        int index = 0;
        int input = 1;

        while (index < difficulty) {
            Coordinates current = emptyCells[index];
            input = 1;

            while (input < difficulty) {
                puzzle[current.getX()][current.getY()] = input;

                if (GameLogic.sudokuIsInvalid(puzzle)) {
                    if (index == 0 && input == GRID_BOUNDARY) {
                        return false;
                    } else if (input == GRID_BOUNDARY) {
                        index--;
                    }
                    input++;
                } else {
                    index++;
                    if (index == difficulty - 1) return true;
                    input = 10;
                }
            }
        }
        return false;
    }
```

Slika 7.3.1. Metoda puzzleIsSolvable

Metoda *puzzleIsSolvable* određuje je li Sudoku igra rješiva. Poziva metodu *typeWriterEnumerate*, koja je objašnjena u daljnjem tekstu, kako bi se dobio niz koordinata praznih polja u mreži. Inicijalizira index 0 i unos (engl. *Input*) 1 te ulazi u while petlju (engl. *While loop*) [47] koja se odrađuje dokle god je index manji od varijable difficulty. Unutar petlje dohvaća prazna polja pomoću *emptyCells* metode klase *Coordinates*, postavlja varijablu input na 1, ulazi u još jednu petlju te nastavlja dokle god je input manji od difficulty. U unutarnjoj petlji postavlja vrijednost trenutnog polja u igri na vrijednost input-a, poziva metodu *sudokuIsInvalid* kako bi se provjerilo odgovara li unos. Ako je netočan unos, ako je index 0 i input je jednak *GRID_BOUNDARY* što je konstanta postavljena na broj 9, vratit će false, ako je input jednak *GRID_BOUNDARY* dekrementirat će se index. Na kraju će se inkrementirati input. Ako je točan unos, inkrementirat će se index, ako je index jednak broju koji je za 1 manji od difficulty-a, vratit će true i postavit će input na 10 kako bi se izašlo iz unutarnje petlje. Vratit će false ako se vanjska petlja izvrši bez pronađenih rezultata. Metoda *typeWriterEnumerate* nabrojat će prazna polja u mreži te će vratiti njihove koordinate. Inicijalizirat će niz tipa *Coordinates* koji je dug jednako varijabli difficulty. Inicijalizira iterator na 0 te prolazi kroz sva polja u mreži. Ako je polje prazno, dodat će koordinate polja u niz praznih ćelija *emptyCells*. Kada iterator postane jednak broju koji je za 1 manji od difficulty-a, vratit će niz praznih ćelija emptyCells. Vraća niz praznih ćelija nakon iteracije kroz cijelu mrežu.

8. ZAKLJUČAK

U ovom radu cilj je bio izraditi *Desktop* aplikaciju za igranje igre Sudoku uz naglasak na funkcionalnost, preglednost i jednostavnost korištenja. Sama igra Sudoku se kroz godine razvijala i rasla te neće zastarjeti kao izazov i kao zabava.

U radu se izrađivala desktop aplikacija koja korisniku omogućuje preglednu i jednostavnu aplikaciju u kojoj mogu riješiti, podijeliti i pogledati prijašnje igre. Uvođenje mogućnosti registracije korisniku donosi dodatne funkcionalnosti kao što su odabir težine igre, nastavak igre te spremanje i dijeljenje rezultata pomoću *Firebase Realtime* baze podataka. Podatci o trenutnoj igri spremaju se lokalno, kao i podatci o registraciji korisnika, dok se podatci o odigranim igrama spremaju u Firebase Realtime bazu podataka kako bi drugi korisnici mogli učitati kod koji će im generirati već odigranu igru koju je netko s njima podijelio.

Iako aplikacija zadovoljava većinu zadanih ciljeva, postoji prostor za proširenje. Jedan od primjera je uvođenje igre s većim dimenzijama mreže što bi uvelo novu vrstu kompleksnosti igre. Još jedno moguće ograničenje aplikacije je lokalno spremanje korisničkih podataka, dok su igre spremljene u bazu podataka što može dovesti do ograničenja određenih funkcionalnosti.

Za razvoj aplikacije mogao se koristiti bilo koji programski jezik i drugo razvojno okruženje s kojima bi dobili slična rješenja s logičke strane. Aplikacija pruža stabilnu osnovu za daljnji razvoj čime se ostavlja prostora za buduća istraživanja i nadogradnje, bilo to u smislu složenosti igre, ili u tehničkom smislu.

LITERATURA

[1] Hrvatska enciklopedija, Sudoku, https://www.enciklopedija.hr/natuknica.aspx?id=58639 [21.08.2024.]

[2] Nezavisne novine, Sudoku https://www.nezavisne.com/zabava/sudoku [21.08.2024.]

[3] Encyclopedia.com, Java https://www.encyclopedia.com/places/asia/indonesian-political-geography/java [21.08.2024.]

[4] Firebase Google documentation, Firebase https://firebase.google.com/docs [21.08.2024.]

[5] Firebase Google documentation, Realtime Database https://firebase.google.com/docs/database [21.08.2024.]

[6] Oracle Help Center, NoSql Database Documentation https://docs.oracle.com/en/database/other-databases/nosql-database/index.html [21.08.2024.]

[7] JSON org, Introducing JSON https://www.json.org/json-en.html [21.08.2024.]

[8] Oracle Help Center, Class String https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html [21.08.2024.]

[9] Oracle Help Center, Introduction to FXML https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html [21.08.2024.]

[10] Oracle Help Center, Defining Methods https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html [21.08.2024.]

[11] Oracle Help Center, Controlling access to Members of a Class https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html [22.08.2024.]

[12] Geeksforgeeks, Singleton Method Design Pattern in Java

https://www.geeksforgeeks.org/singleton-class-java/ [23.08.2024.]

[13] Oracle Help Center, Creating New Class Instances

https://docs.oracle.com/javase/tutorial/reflect/member/ctorInstance.html [24.08.2024.]

[14] Geeksforgeeks. What are Objects in Programming?

https://www.geeksforgeeks.org/what-are-objects-in-programming/ [28.08.2024.]

[15] Oracle Help Center, Class File

https://docs.oracle.com/javase/8/docs/api/java/io/File.html [28.08.2024.]

[16] Oracle Help Center, Map

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html [28.08.2024.]

[17] Oracle Help Center, Chapter 3 Lexical Structure

https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.1 [28.08.2024.]

[18] Geeksforgeeks, Serialization and Deserialization in Java with Example

https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.1 [28.08.2024.]

[19] Oracle Help Center, Java Overview

https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Java-overview.html#GUID-68EE1A7B-1F78-4074-AB76-AF9B2CE878F6 [28.08.2024.]

[20] Geeksforgeeks, Getter and Setter in Java

https://www.geeksforgeeks.org/getter-and-setter-in-java/ [28.08.2024.]

[21] TutorialsPoint, Jackson Annotations - @JsonCreator

https://www.tutorialspoint.com/jackson_annotations/jackson_annotations_jsoncreator [28.08.2024.]

[22] Oracle Help Center, Annotations Basics

https://docs.oracle.com/javase/tutorial/java/annotations/basics.html [28.08.2024]

[23] Oracle Help Center, Enum Types

https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html [29.08.2024.]

[24] Geeksforgeeks, What is a Pixel?

https://www.geeksforgeeks.org/what-is-a-pixel/ [29.08.2024.]

[25] Geeksforgeeks, Java Constructors

https://www.geeksforgeeks.org/constructors-in-java/ [30.08.2024.]

[26] Javatpoint, What is an Instance in Java

https://www.javatpoint.com/what-is-an-instance-in-java [30.08.2024.]

[27] Oracle Help Center, Class ActionEvent

https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionEvent.html [30.08.2024.]

[28] Oracle Help Center, The switch Statement

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html [31.08.2024.]

[29] Oracle Help Center, Abstract Methods and Classes

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html [01.09.2024.]

[30] Amazon AWS Documentation, Asynchronous Programming

https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/basics-async [01.09.2024.]

[31] Britannica, URL

https://www.britannica.com/technology/URL [01.09.2024]

[32] Mongodb, what is NoSQL

https://www.mongodb.com/resources/basics/databases/nosql-explained [03.09.2024.]

[33] Oracle Help Center, Class Hashmap

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html [03.09.2024.]

[34] Geeksforgeeks, Object toString() Method in Java

https://www.geeksforgeeks.org/object-tostring-method-in-java/ [03.09.2024.]

[35] Clourflare, What is HTTP?

https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/ [03.09.2024.]

[36] Geeksforgeeks, Different kinds of HTTP requests

https://www.geeksforgeeks.org/different-kinds-of-http-requests/ [04.09.2024.]

[37] Javadoc, Class ObjectMapper

https://javadoc.io/doc/com.fasterxml.jackson.core/jackson-databind/2.3.1/com/fasterxml/jackson/databind/ObjectMapper.html [04.09.2024.]

[38] Mdn web docs, HTTP response status codes

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status [05.09.2024.]

[39] Oracle Help Center, Class BufferedReader

https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html [06.09.2024.]

[40] Oracle Help Center, Class StringBuilder

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html [06.09.2024.]

[41] Oracle Help Center, Introduction to Event Listeners

https://docs.oracle.com/javase%2Ftutorial%2F/uiswing/events/intro.html [07.09.2024.]

[42] Oracle Help Center, Class UUID

https://docs.oracle.com/javase/8/docs/api/java/util/UUID.html [07.09.2024.]

[43] Geeksforgeeks, Java.util.Collections.frequency() in Java

https://www.geeksforgeeks.org/java-util-collections-frequency-java/ [07.09.2024.]

[44] Oracle Help Center, Class Random

https://docs.oracle.com/javase/8/docs/api/java/util/Random.html [07.09.2024.]

[45] Cornell University Department of Mathematics, The Math Behind Sudoku

https://pi.math.cornell.edu/~mec/Summer2009/Mahmood/Symmetry.html [07.09.2024.]

[46] Cornell University Department of Mathematics, Mathematics and Sudokus: Solving Algorithms (II) Crook's pencil-and-paper algorithm

https://pi.math.cornell.edu/~mec/Summer2009/meerkamp/Site/Solving_any_Sudoku_II.html [07.09.2024.]

[47] Oracle Help Center, The while and do-while Statements

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html [07.09.2024.]

SAŽETAK

U ovome radu napravljena je *desktop* aplikacija za igranje logičke igre sudoku. Programski jezik korišten za izradu aplikacije je Java u programskom okruženju Intellij. Korištena je *NoSQL Firebase Realtime Database* za spremanje podataka. Opisani su zahtjevi za aplikaciju te implementacija funkcionalnosti u aplikaciji. Aplikacija omogućuje korisniku registraciju ili igru kao gost, generiranje sudoku mreže sa različitim težinama, nastavak postojeće igre, prikaz rezultata 10 zadnjih odigranih igara te slanje koda igre putem e-mail kanala i generiranje igre iz koda.

Ključne riječi: desktop aplikacija, Intellij, Java, Sudoku

ABSTRACT

Title: Desktop application for Sudoku

In this paper a desktop application for playing logic game sudoku has been created. Programming language used for creating the application is Java, in Intellij integrated development environment. A NoSQL Firebase Realtime Database has been used for data storage. Application requirements have been described as well as the application functionality implementation. The application allows the user to register or play as a guest, allows generating a sudoku grid with different difficulties, continuing an existing game, displaying results of 10 last played games, sending a game code via an e-mail channel and generating a game from the code.

Key words: desktop application, Intellij, Java, Sudoku

PRILOG

[1] Programski kod [online] dostupno na: https://gitlab.com/karlov_projekt/sudoku-desktop-application

ŽIVOTOPIS

Karlo Madžarević rođen je 24.03.2001. godine u Slavonskom Brodu u Hrvatskoj. Pohađao je Osnovnu školu Stjepan Radić u Oprisavcima. Nakon završene osnovne škole upisuje Tehničku školu Slavonski Brod za smjer elektrotehničara. Za vrijeme školovanja bavi se rukometom i sudjeluje na državnim prvenstvima. Maturirao je 2019. godine te 2020. upisuje Stručni studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. U siječnju 2023. započinje karijeru kao Java Spring Boot Backend developer. U Veljači 2024. prelazi na poziciju Junior Fullstack developera koristeći tehnologije Kotlin Spring Boot i React.

_____

Potpis autora