

Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Evolučný výskum využitia AI k vytvoreniu funkčného programu

Semestrálna práca pre predmet SWI

David Balus, Marko Kožiarik, Filip Rosa

Ostrava, apríl 2025

Obsah

1	Začíname	5
1.1	Prvý prompt	5
1.2	Pridanie popisu systému	6
1.3	Pridanie požiadaviek	11
1.4	Prvý problém	21
2	Use Case	23
2.1	Dosiahol sa tvoj limit tarify Free pre GPT-4o	24
2.2	Pridanie Use Case scenárov	35
2.3	Výsledok	41
3	Class diagram	42
3.1	Class, class, interface...	43
3.2	Ukladáme dáta	52
3.3	Ono to nesype	59
3.4	Problém za problémom	84
4	Malými krôčkami k výsledku	89
4.1	Spájame funkcionality	89
5	Záver	121

Zoznam obrázkov

1	Príkaz s nastolením problému	5
2	Odpoveď ChatGPT na príkaz s nastolením problému	5
3	Príkaz s popisom systému	6
4	Odpoveď ChatGPT na príkaz s popisom systému	7
5	Príkaz s FURPS+ požiadavkami	11
6	Odpoveď ChatGPT na príkaz s FURPS+ požiadavkami	12
7	Návrh UI	12
8	Príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami	13
9	Odpoveď ChatGPT na príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami	13
10	Odpoveď ChatGPT na príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami - ďalšie fázy	16
11	Odpoveď ChatGPT na príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami - databáza	21
12	Oprava AAPT erroru	22
13	Stav aplikácie po prvej kapitole	23
14	Príkaz s popismi Use Cases	24
15	Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 1	24
16	Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 2	26
17	Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 3	27
18	Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 4	28
19	Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 5 a 6	29
20	Stav aplikácie po pridaní popisu Use Cases	35
21	Príkaz s popismi scenárov Use Cases	36
22	Odpoveď ChatGPT na príkaz s popismi scenárov Use Cases	37
23	Stav aplikácie po druhej kapitole	41
24	Class diagram	42
25	Príkaz s PlantUML reprezentáciou Class diagramu	45
26	Odpoveď ChatGPT na príkaz s PlantUML reprezentáciou Class diagramu	46
27	Príkaz pre začatie používania Room databázy	52
28	Odpoveď ChatGPT na príkaz pre začatie používania Room databázy . . .	52
29	Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 1.krok	52
30	Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 2.krok	55
31	Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 3.krok	56
32	Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 4.krok	57
33	Príkaz pre zlúčenie dvoch kódov do jedného	59
34	Odpoveď ChatGPT na príkaz pre zlúčenie dvoch kódov do jedného	60
35	Ukážka z logu Android Studia o páde aplikácie	84

Listings

1	User.java	8
2	ParkingSpot.java	8
3	Reservation.java	9
4	LoginActivity.java	10
5	LoginActivity.java	14
6	activity_login.xml	15
7	ParkingSpot.java	16
8	MainActivity.java	17
9	activity_main.xml	18
10	ReservationActivity.java	18
11	ParkingSpotAdapter.java	19
12	activity_reservation.xml	20
13	AAPT error	21
14	MainActivity.java	25
15	ReservationActivity.java	26
16	ReservationListActivity.java	27
17	ReservationListActivity.java - extended	28
18	SettingsActivity.java	29
19	activity_login.xml	30
20	activity_reservation.xml	31
21	activity_reservation_list.xml	31
22	activity_settings.xml	32
23	activity_parking_map.xml	32
24	item_parking_spot.xml	33
25	dialog_cancel_reservation.xml	34
26	activity_login.xml	38
27	activity_parking_map.xml	39
28	activity_reservation.xml	39
29	activity_reservation.xml	40
30	PlantUML reprezentácia Class diagramu	43
31	Authenticable.java	47
32	SSOAuthentication.java	47
33	CodeAuthentication.java	48
34	User.java	49
35	ParkingSystem.java	50
36	Reservation.java	50
37	ParkingSpot.java	51
38	ParkingLot.java	51
39	User.java	53
40	Reservation.java	53
41	ParkingSpot.java	54
42	ParkingLot.java	54
43	UserDAO.java	55
44	ReservationDAO.java	55
45	ParkingSpotDao.java	56
46	AppDatabase.java	56
47	AuthActivity.java	57

48	ReservationActivity.java	58
49	AppDatabase.java	60
50	User.java	61
51	UserDAO.java	63
52	SSOAuthentication.java	64
53	SettingsActivity.java	65
54	Reservation.java	66
55	Authenticable.java	67
56	ReservationActivity.java	68
57	ReservationListActivity.java	70
58	ReservationDAO.java	72
59	ParkingSpot.java	73
60	ParkingSpotDAO.java	74
61	ParkingSpotAdapter.java	75
62	ParkingLot.java	76
63	ParkingSystem.java	77
64	LoginActivity.java	78
65	CodeAuthentication.java	80
66	CodeDAO.java	81
67	Converters.java	81
68	MainActivity.java	82
69	MainActivity.java	85
70	ReservationActivity.java	87
71	AndroidManifest.xml	89
72	AppDatabase.java	90
73	Authenticable.java	91
74	CodeAuthentication.java	92
75	CodeDAO.java	93
76	Converters.java	93
77	LoginActivity.java	94
78	MainActivity.java	96
79	MapActivity.java	98
80	ParkingLot.java	102
81	ParkingSpot.java	103
82	ParkingSpotAdapter.java	105
83	ParkingSpotDAO.java	106
84	ParkingSystem.java	107
85	Reservation.java	108
86	ReservationActivity.java	110
87	ReservationAdapter.java	112
88	ReservationDAO.java	113
89	ReservationListActivity.java	114
90	SettingsActivity.java	116
91	SSOAuthentication.java	117
92	User.java	118
93	UserDAO.java	120

1 Začíname

Počas tvorby nášho semestrálneho projektu **Systém pre správu parkoviska VŠB-TUO** do predmetu Úvod do softwarového inžinýrství nám bolo v polovici semestra zadané doplňujúce, dobrovoľné rozšírenie projektu o vypracovanie tejto štúdie, v rámci ktorej sme sa mali zamerať na vytvorenie funkčnej podoby nášho projektu.

Výzvu sme teda prijali. Cieľom je za pomoci AI, konkrétne ChatGPT, podrobnej dokumentácie jednotlivých promptov a využitia našej hlavnej časti projektu - UML diagramov, vytvoriť funkčnú aplikáciu. Nakoľko nikto z náš nemá skúsenosti s vývojom aplikácií pre iOS, vytvárali sme aplikáciu pre Android s využitím Android Studia a programovacieho jazyku Java.

Viac informácií o našom projekte je možné nájsť na jeho webe.

1.1 Prvý prompt

Začali sme zľahka, a najprv sme zistili, či nám vôbec bude ChatGPT ochotný pomôcť.

Zdravím. Začínam vytvárať jeden veľmi významný projekt - Systém pre správu parkoviska VŠB-TUO a budem potrebovať tvoju pomoc s nakódením danej aplikácie. Programovať budem pre Android, v Android Studiu a v jazyku Java. Postupne ti sem budem písať rôzne popisy systému a UML diagramy, ktoré budú popisovať funkčnosť a prezentáciu danej aplikácie. Čo ty na to, si ochotný mi pomôcť?

Obr. 1: Príkaz s nastolením problému

Na danú otázku sme dostali (pomerne neprekvapivo) kladnú odpoveď.

Ahoj! Jasně, velmi rád ti s tím pomůžem – zní to jako zajímavý a užitečný projekt. 🧑🏻💻🚗

Android Studio + Java je úplně v pohodě kombinácia, a ak mi sem budeš postupne dávať popisy, UML diagramy alebo konkrétne požiadavky na funkcionality, môžeme to spolu krok po kroku vybudovať. Klúadne začni prvým popisom systému, alebo diagramom – prispôbím sa tomu, čo práve potrebuješ.

Pohodlne sa usad', a poďme na to! 💪

Obr. 2: Odpoveď ChatGPT na príkaz s nastolením problému

1.2 Pridanie popisu systému

ChatGPT od nás dostal popis nášho systému s otázkou, nech sa pokúsi na základe neho niečo vygenerovať.

Tu je popis systému - Systém bude mobilná aplikácia určená špecificky pre VŠB-TUO, ktorá bude dostupná pre zariadenia s operačnými systémami Android aj iOS. Používať ju budú môcť iba študenti a zamestnanci VŠB prostredníctvom prihlásenia cez SSO, pričom návštevníci budú mať možnosť jednorazového prístupu pomocou kódu. Parkoviská budú rozdelené podľa kategórií používateľov, napríklad pre zamestnancov a študentov. Rezervácie budú viazané na meno a prihlasovacie údaje.

Hlavné funkcie aplikácie zahŕňajú možnosť rezervácie iba jedného parkovacieho miesta, zrušenie rezervácie najneskôr 30 minút pred jej začiatkom, výber časového intervalu rezervácie a náhľad mapy s aktuálnym zobrazením dostupnosti parkovacích miest.

Dokážeš na jeho základe vytvoriť nejaký kód?

Obr. 3: Príkaz s popisom systému

Ako odpoveď sme dostali naprv pekné zhrnutie požiadaviek formou nečíslovaného zoznamu. Potom nám bol vygenerovaný návrh prvého kroku, ktorý nám vytvoril spustiteľný kód v podobe zopár základných deklarácií tried, ktoré sme následne vložili do Android Studia.

Samozrejme, nedostali sme žiaden viditeľný výsledok, nakoľko dané triedy nemajú svoju implementáciu.

Perfektný úvod, ďakujem za detailný popis! Áno, vieme na tom okamžite začať pracovať – môžeme začať budovať aplikáciu po častiach.

Zhrnutie požiadaviek:

- Platforma: Android (aj iOS – ale sústredíme sa najprv na Android/Java)
 - Prihlásenie: SSO pre študentov/zamestnancov, jednorazový kód pre návštevníkov
 - Rezervácia parkovacích miest:
 - Len jedno miesto naraz
 - Možnosť zrušiť najneskôr 30 minút pred začiatkom
 - Výber časového intervalu
 - Mapa so zobrazením dostupnosti
 - Parkoviská sú rozdelené podľa typu používateľa
-

Návrh prvého kroku:

Navrhujem začať týmto:

1. **Struktúra projektu:** Vytvorenie základných aktivít a modelov.
2. **Autentifikácia:** Pridáme jednoduchú login aktivitu (SSO sa bude integrovať neskôr).
3. **Doménové triedy (Modely):** Vytvoríme si základné entity ako `User`, `ParkingSpot`, `Reservation`.
4. **Rezervačný systém (základ):** Základná logika, že si môžeš rezervovať miesto.

Obr. 4: Odpoveď ChatGPT na príkaz s popisom systému

Vytvorené nám boli štyri triedy - `User.java`, `Reservation.java`, `ParkingSpot.java` a `LoginActivity.java`. Nižšie sú uvedené ich zdrojové kódy.


```

public class User {
    private String id;
    private String name;
    private String role; // "student", "employee", "visitor"

    public User(String id, String name, String role) {
        this.id = id;
        this.name = name;
        this.role = role;
    }

    // Gettery a settery
    public String getId() { return id; }
    public String getName() { return name; }
    public String getRole() { return role; }
}

```

Listing 1: User.java

```

public class ParkingSpot {
    private String id;
    private boolean isAvailable;
    private String category; // "student", "employee"

    public ParkingSpot(String id, boolean isAvailable, String
        category) {
        this.id = id;
        this.isAvailable = isAvailable;
        this.category = category;
    }

    public String getId() { return id; }
    public boolean isAvailable() { return isAvailable; }
    public String getCategory() { return category; }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }
}

```

Listing 2: ParkingSpot.java

```

import java.time.LocalDateTime;

public class Reservation {
    private String id;
    private String userId;
    private String parkingSpotId;
    private LocalDateTime startTime;
    private LocalDateTime endTime;

    public Reservation(String id, String userId, String parkingSpotId
        , LocalDateTime startTime, LocalDateTime endTime) {
        this.id = id;
        this.userId = userId;
        this.parkingSpotId = parkingSpotId;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    // Gettery a logika
    public boolean canBeCancelled() {
        return LocalDateTime.now().isBefore(startTime.minusMinutes
            (30));
    }
}

```

Listing 3: Reservation.java

```

public class LoginActivity extends AppCompatActivity {
    private EditText nameInput;
    private Button loginButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        nameInput = findViewById(R.id.nameInput);
        loginButton = findViewById(R.id.loginButton);

        loginButton.setOnClickListener(v -> {
            String name = nameInput.getText().toString();
            User user = new User("1", name, "student"); // docasne
                hardcoded
            // Prechod do hlavnej aktivity
            Intent intent = new Intent(LoginActivity.this,
                MainActivity.class);
            intent.putExtra("userName", user.getName());
            startActivity(intent);
        });
    }
}

```

Listing 4: LoginActivity.java

1.3 Pridanie požiadaviek

Ako ďalší krok nasledovalo predanie **F** (functionality) **U** (usability) **R** (reliability) **P** (performance) **S** (supportability) + požiadaviek. Taktiež sme od neho chceli už aj nejaký základný návrh user interface.

Teraz ti pridám funkčné, nefunkčné a FURPS+ požiadavky.

Funkčné požiadavky

- Prihlásenie používateľa cez SSO
- Požiadanie o jednorazový prihlasovací kód
- Náhľad na parkoviská a ich voľné miesta
- Rezervácia parkovacieho miesta
- Zrušenie rezervácie
- Zmena času/miesta rezervácie (ak je to možné)
- Odhlásenie

Nefunkčné požiadavky

- Mobilná aplikácia bude multiplatformová (Android, iOS)
- Používateľské rozhranie bude intuitívne, prehľadné a jednoduché na používanie
- Systém odpovie používateľom do 5 sekúnd
- Aplikácia musí zvládnuť až 1000 aktívnych prihlásených používateľov bez výpadkov
- Systém bude automaticky zálohovať aktuálne a budúce rezervácie každých 24 hodín

FURPS+

- Funkcionalita / Functionality
 - Aplikácia umožní používateľom prihlásenie cez SSO
 - Systém bude podporovať jazyky CZ a ENG (nie SK, sorry chlapci)
 - Aplikácia umožní prihláseným používateľom náhľady a rezervácie parkovacích miest v reálnom čase
- Použitelnosť / Usability
 - Rozhranie bude intuitívne, s možnosťou prepínania medzi tmavým a svetlým režimom
 - Voľné a obsadené parkovacie miesta budú zobrazované prehľadným spôsobom
- Spoľahlivosť / Reliability
 - Systém bude dostupný 99 % času (každodenná záloha o polnoci)
 - Server zvládne až 1000 aktívnych prihlásených používateľov bez výpadkov
- Výkonnosť / Performance
 - Odozva systému do 5 sekúnd
 - Načítanie stránky do 3 sekúnd
- Podpora a údržba / Supportability
 - Kompatibilita s Android a iOS operačnými systémami
 - Systém a softvér budú podrobne zdokumentované vrátane príslušných diagramov

Dokážeš vytvoriť zároveň aj nejakú vizuálnu reprezentáciu v podobe UI?



Obr. 5: Príkaz s FURPS+ požiadavkami

Pri odpovedi sme si všimli, *okrem toho ako sme úžasní*, nejasne zadanú otázku. Očakávali sme totiž vytvorenie UI v podobe kódu, dostali sme však obrázok.

Perfektne spracované požiadavky! 🥳 Máš to krásne zoradené, takže teraz sa dá systém budovať presne podľa toho, čo potrebuješ. A áno — môžem ti pripraviť aj vizuálne návrhy používateľského rozhrania (UI) podľa týchto požiadaviek.

🔍 Zhrnutie toho, čo teraz navrhmem:

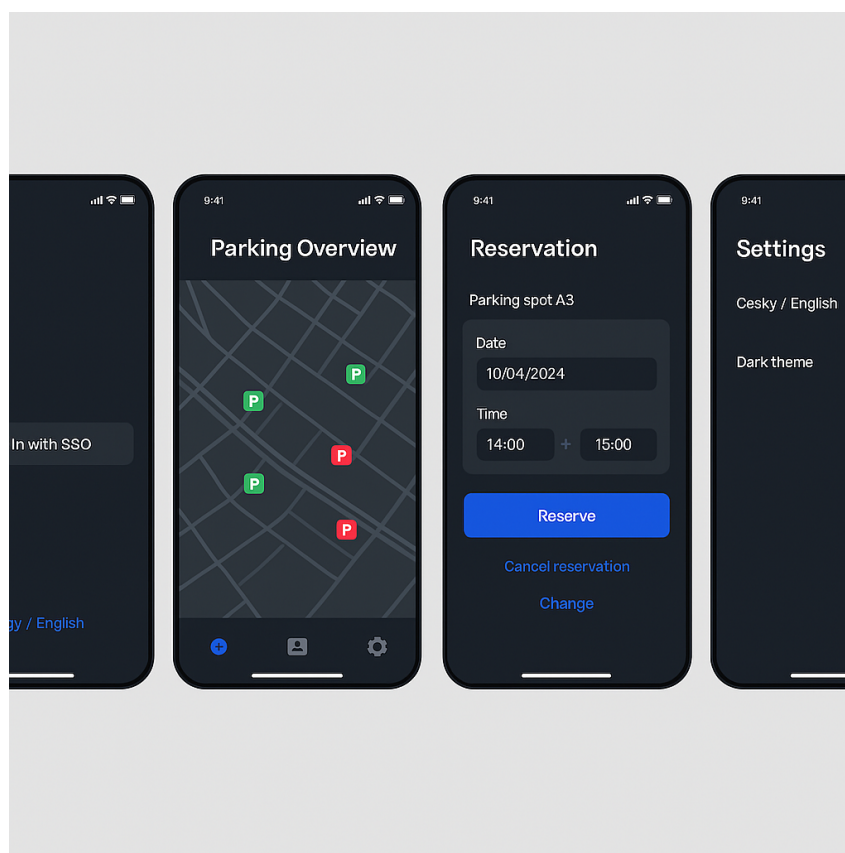
Na základe požiadaviek pripravím vizuálny návrh (UI wireframe) pre nasledujúce obrazovky:

1. Login obrazovka (SSO / jednorazový kód)
2. Prehľad parkovísk s vizualizáciou voľných/obsadených miest
3. Detail rezervácie (rezervovať, zmeniť, zrušiť)
4. Nastavenia (prepnutie jazyka a tmavý/svetlý režim)

🎨 Vizuálny štýl podľa FURPS+:

- Jednoduché a intuitívne UI
- Podpora CZ/ENG (texty budú označené ako prepínateľné)
- Tmavý/svetlý režim (switch v nastaveniach)
- Zobrazenie parkovacích miest prehľadnou formou – farby a ikony

Obr. 6: Odpoveď ChatGPT na príkaz s FURPS+ požiadavkami



Obr. 7: Návrh UI

Preto sme ho následne, samozrejme slušne, požiadali o kód.

Ja by som poprosil hlavne generovať kód k aplikácii. :)

Obr. 8: Príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami

Dostali sme jasnú odpoveď, ide sa kódovať!

Rozumiem, ideme rovno na **kódovanie!** 🤖💻

Vytvorím ti teraz základ **funkčného kostra systému podľa požiadaviek**. Začneme tým, čo sme už spomínali a čo patrí medzi najdôležitejšie **funkčné požiadavky**:

📌 **FÁZA 1: Prihlásenie používateľa cez SSO / jednorazový kód**

Keďže SSO je špecifický systém (pravdepodobne cez Shibboleth/VŠB infraštruktúru), zatiaľ to simulujeme ako výber medzi 2 možnosťami:

- **SSO Login (študent/zamestnanec)**
- **Jednorazový kód (návštevník)**

Obr. 9: Odpoveď ChatGPT na príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami

ChatGPT našu požiadavku pre generovanie kódu rozdelil na štyri fázy. Prvá fáza bola automaticky vygenerovaná a obsahovala kód pre prihlásenie používateľa cez SSO, prípadne pomocou jednorazového kódu. Trieda LoginActivity bola aktualizovaná, a bolo nám k nej vytvorené aj príslušné XML.

```

public class LoginActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;
    private EditText inputVisitorCode;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);
        inputVisitorCode = findViewById(R.id.inputVisitorCode);

        btnSSOLogin.setOnClickListener(v -> {
            // Simulovane SSO prihlasenie
            User user = new User("123", "Jan Novak", "student");
            goToMainScreen(user);
        });

        btnVisitorLogin.setOnClickListener(v -> {
            String code = inputVisitorCode.getText().toString();
            if (code.equals("123456")) {
                User user = new User("visitor001", "Visitor", "visitor");
                goToMainScreen(user);
            } else {
                Toast.makeText(this, "Neplatny kod", Toast.LENGTH_SHORT).show();
            }
        });
    }

    private void goToMainScreen(User user) {
        Intent intent = new Intent(LoginActivity.this, MainActivity.class);
        intent.putExtra("userName", user.getName());
        intent.putExtra("userRole", user.getRole());
        startActivity(intent);
        finish();
    }
}

```

Listing 5: LoginActivity.java

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="24dp"
    android:gravity="center"
    android:background="#f5f5f5">

    <Button
        android:id="@+id/btnSSOLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prihlasit sa cez SS0" />

    <EditText
        android:id="@+id/inputVisitorCode"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Jednorazovy kod"
        android:inputType="number"
        android:layout_marginTop="16dp" />

    <Button
        android:id="@+id/btnVisitorLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prihlasit ako navstevnik"
        android:layout_marginTop="8dp" />
</LinearLayout>

```

Listing 6: activity_login.xml

Následne nám bola ponúknutá možnosť vytvoriť kód aj pre ostatné fázy, konkrétne prehľad parkovacích miest a ich dostupnosť, rezervácia a zrušenie alebo úprava rezervácie. Túto možnosť sme samozrejme využili.

✓ Ďalšie kroky:

Keďže máme prihlásenie ako **funkčný základ**, môžeme pokračovať:

- FÁZA 2: Prehľad parkovacích miest + ich dostupnosť
- FÁZA 3: Rezervácia (1 miesto, výber času)
- FÁZA 4: Zrušenie alebo úprava rezervácie (ak je možné)

👉 Povedz, ktorú časť chceš ako ďalšiu (napr. zoznam parkovísk s dostupnosťou), a ideme ďalej písať kód!

Obr. 10: Odpoveď ChatGPT na príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami - ďalšie fázy

Dostali sme triedy rozšírenie ParkingSpot.java, MainActivity.java, ParkingSpotAdapter.java, ReservationActivity.java a ich príslušné XML reprezentácie UI.

```
public class ParkingSpot {
    private String id;
    private String label;
    private boolean isAvailable;
    private String category; // student, employee

    public ParkingSpot(String id, String label, boolean isAvailable,
        String category) {
        this.id = id;
        this.label = label;
        this.isAvailable = isAvailable;
        this.category = category;
    }

    public String getId() { return id; }
    public String getLabel() { return label; }
    public boolean isAvailable() { return isAvailable; }
    public String getCategory() { return category; }

    public void setAvailable(boolean available) { this.isAvailable =
        available; }
}
```

Listing 7: ParkingSpot.java

```

public class MainActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private ParkingSpotAdapter adapter;
    private List<ParkingSpot> allSpots = new ArrayList<>();
    private String userRole;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        String userName = getIntent().getStringExtra("userName");
        userRole = getIntent().getStringExtra("userRole");

        setTitle("Vitaj, " + userName);

        loadDummySpots(); // neskôr nahradíme databázou

        adapter = new ParkingSpotAdapter(getFilteredSpots(), this::
            onSpotSelected);
        recyclerView.setAdapter(adapter);
    }

    private void loadDummySpots() {
        allSpots.add(new ParkingSpot("1", "A1", true, "student"));
        allSpots.add(new ParkingSpot("2", "A2", false, "student"));
        allSpots.add(new ParkingSpot("3", "B1", true, "employee"));
        allSpots.add(new ParkingSpot("4", "B2", true, "employee"));
    }

    private List<ParkingSpot> getFilteredSpots() {
        List<ParkingSpot> filtered = new ArrayList<>();
        for (ParkingSpot spot : allSpots) {
            if (userRole.equals("visitor") || spot.getCategory().equals(
                userRole)) {
                filtered.add(spot);
            }
        }
        return filtered;
    }

    private void onSpotSelected(ParkingSpot spot) {
        Intent intent = new Intent(this, ReservationActivity.class);
        intent.putExtra("spotId", spot.getId());
        intent.putExtra("spotLabel", spot.getLabel());
        startActivity(intent);
    }
}

```

Listing 8: MainActivity.java

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>

```

Listing 9: activity_main.xml

```

public class ReservationActivity extends AppCompatActivity {

    private TextView spotLabel;
    private Button btnReserve;
    private TimePicker timePicker;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation);

        spotLabel = findViewById(R.id.spotLabel);
        timePicker = findViewById(R.id.timePicker);
        btnReserve = findViewById(R.id.btnReserve);

        String spotId = getIntent().getStringExtra("spotId");
        String label = getIntent().getStringExtra("spotLabel");

        spotLabel.setText("Rezervacia miesta " + label);

        btnReserve.setOnClickListener(v -> {
            int hour = timePicker.getHour();
            int minute = timePicker.getMinute();

            // Tu by bola rezervacia ulozena (do databazy / backendu)
            Toast.makeText(this, "Miesto " + label + " rezervovane na
                " + hour + ":" + String.format("%02d", minute), Toast
                .LENGTH_SHORT).show();
            finish();
        });
    }
}

```

Listing 10: ReservationActivity.java

```

public class ParkingSpotAdapter extends RecyclerView.Adapter<
    ParkingSpotAdapter.ViewHolder> {

    private List<ParkingSpot> spots;
    private OnSpotClickListener listener;

    public interface OnSpotClickListener {
        void onSpotClick(ParkingSpot spot);
    }

    public ParkingSpotAdapter(List<ParkingSpot> spots, OnSpotClickListener
        listener) {
        this.spots = spots;
        this.listener = listener;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
        viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(android.R.layout.simple_list_item_2, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position)
    {
        ParkingSpot spot = spots.get(position);
        holder.label.setText("Miesto: " + spot.getLabel());
        holder.status.setText(spot.isAvailable() ? "Volne" : "Obsadene");
        holder.itemView.setOnClickListener(v -> listener.onSpotClick(spot))
            ;
    }

    @Override
    public int getItemCount() {
        return spots.size();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView label, status;

        public ViewHolder(@NonNull View itemView) {
            super(itemView);
            label = itemView.findViewById(android.R.id.text1);
            status = itemView.findViewById(android.R.id.text2);
        }
    }
}

```

Listing 11: ParkingSpotAdapter.java

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="24dp">

    <TextView
        android:id="@+id/spotLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Miesto"
        android:textSize="20sp"
        android:layout_marginBottom="16dp" />

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:is24HourView="true" />

    <Button
        android:id="@+id/btnReserve"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Rezervovat"
        android:layout_marginTop="24dp" />
</LinearLayout>

```

Listing 12: activity_reservation.xml

Na konci celého „procesu“ nám bola ponúknutá možnosť integrovať backend alebo aspoň lokálnu databázu. Povedali sme si prečo nie, **to sme však ešte nevedeli do akých problémov sa dostávame a ako nám toto rozhodnutie skomplikuje život**, a vybrali sme si Room databázu.

Room databáza je knižnica pre Android poskytovaná spoločnosťou Google, ktorá zjednodušuje prácu s databázami SQLite. Slúži ako abstrakcia nad SQLite, čo uľahčuje spracovanie perzistencie údajov v aplikáciách pre Android a zároveň zabezpečuje overenie dotazov počas kompilácie a znižuje štandardný kód.






Zatiaľ sme nepožadovali vygenerovať kód preňu, no chceli sme, nech v budúcnosti s tou databázou počíta.

FÁZA 4: Zrušenie / zmena rezervácie

Zatiaľ to môžeme simulovať ako **jednoduchý prehľad s možnosťou "Zrušiť"**. Skutočná zmena rezervácie by vyžadovala backend alebo aspoň lokálnu databázu (napr. Room alebo Firebase). Ale môžeme pripraviť simuláciu a výstupný dialóg.

Chceš, aby som to doprogramoval hneď teraz, alebo máš záujem rovno integrovať databázu (napr. Room)?

Sumár:

-  Login (SSO / kód)
-  Výber parkoviska
-  Výber času a rezervácia
-   Zrušenie / zmena (potrebuje stav rezervácií)

Daj vedieť, či pokračujeme rovno na zrušenie / úpravu rezervácie alebo chceš ísť do **Room databázy / Firebase** a uložiť reálne údaje.

Obr. 11: Odpoveď ChatGPT na príkaz s rozširujúcou požiadavkou na kód k príkazu s FURPS+ požiadavkami - databáza

1.4 Prvý problém

Akonáhle sme všetko pekne nakopírovali do Android Studia, pri kompilácii sme narazili na prvý problém.

```
ERROR: AndroidStudioProjects\ParkingVSB\ParkingVSB\app\src\main\res\
layout\activity_reservation.xml:24: AAPT: error: attribute android:
    is24HourView not found.
```

Listing 13: AAPT error

Dostali sme však opravu, ktorá je funkčná a podarilo sa nám projekt úspešne skompilovať bez erroru.

Riešenie

✓ **Možnosť 1: Nastav** `setIs24HourView(true)` **v Java**

Odstráň tento riadok v XML:

xml

 Kopírovať

 Upraviť

```
android:is24HourView="true" ✖
```

A namiesto toho pridaj do `ReservationActivity.java` po `findViewById()` toto:

java

 Kopírovať

 Upraviť

```
timePicker.setIs24HourView(true);
```

Obr. 12: Oprava AAPT erroru

2 Use Case

V prvej kapitole nám ChatGPT vygeneroval niečo, čo dalo skompilovať a buildnúť. Síce až na x-tý pokus, ale buildlo sa to. Avšak jediné, čo sa zobrazí na telefóne je biela obrazovka. Teda biela, ono je v jemnom odtieni fialovej...

V tejto kapitole predáme ChatGPT náš Use Case diagram, popis jednotlivých Use Cases a ich scenáre. Pokúsime sa zároveň z neho vytiahnuť niečo, na čo pôjde klikáť.



Obr. 13: Stav aplikácie po prvej kapitole

2.1 Dosiahol sa tvoj limit tarify Free pre GPT-4o

Problémy, problémy, problémy... Nakoľko sme dosiahli limit pre model GPT-4o, a zároveň odmietame platiť za ChatGPT 20\$ mesačne, command s obrázkom Use Casu sme museli preskočiť. Prešli sme preto rovno na popis Use Cases.

Pokús sa následne vylepšiť daný kód na základe popisu use case diagramu. Taktiež by som rád, aby sa po zapnutí aplikácie už dalo na niečo klikat'.

Use Case 1 - Prihlásenie užívateľa

- Popis: Tento use case umožní užívateľom prihlásiť sa do systému.
- Primárny aktér: Študent alebo zamestnanec VŠB-TUO, návštevník
- Zainteresované strany a záujmy: Užívatelia potrebujú prístup k rezervácii parkovacích miest.
- Predpoklad: Študent alebo zamestnanec má prístup k SSO autentifikácii. Návštevník obdržal jednorazový kód.
- Spúšťač: Užívateľ stlačí tlačidlo "Prihlásiť sa cez SSO" alebo "Prihlásiť jednorazovým kódom".
- Minimálne záruky: Systém vypíše výsledok prihlásenia - úspech/neúspech.
- Záruky úspechu: Systém umožní užívateľovi rezervovať si parkovacie miesto.
- Hlavný scénar: 1. Užívateľ otvorí mobilnú aplikáciu.
2. Aplikácia zobrazí prihlasovaciu obrazovku.
3. Užívateľ vyberie ↓ ožnosť "Prihlásiť sa cez SSO" alebo "Prihlásiť sa

Obr. 14: Príkaz s popismi Use Cases

Začali sme s Use Case 1 pre prihlásenie užívateľa. Ten obsahoval kód MainActivity.java.

1. Prihlásenie užívateľa (Use Case 1)

Popis:

- Užívateľ sa prihlási buď pomocou SSO (Single Sign-On) alebo jednorazového kódu.

Vylepšenie:

- **SSO autentifikácia:** Implementovať možnosť prihlásenia cez SSO pre študentov a zamestnancov.
- **Jednorazový kód:** Implementovať zadanie jednorazového kódu pre návštevníkov.

Implementácia:

1. **Prihlasovacia obrazovka:** Máme obrazovku s tlačidlami na prihlásenie cez SSO alebo s jednorazovým kódom. Po kliknutí na tlačidlo by mal systém presmerovať užívateľa na požadovanú prihlasovaciu metódu.
2. **Validácia údajov:** Ak údaje nie sú správne, zobrazit' chybovú správu.
3. **Zobrazenie hlavného rozhrania:** Po úspešnom prihlásení sa užívateľ dostane na hlavnú obrazovku aplikácie, kde si môže rezervovať parkovacie miesto.

Obr. 15: Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 1

```

public class MainActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login); // Assuming
            activity_login.xml is the login screen

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);

        btnSSOLogin.setOnClickListener(v -> loginSSO());
        btnVisitorLogin.setOnClickListener(v -> loginVisitor());
    }

    private void loginSSO() {
        // Handle SSO login process
        // Redirect to SSO authentication screen
    }

    private void loginVisitor() {
        // Handle visitor login with one-time code
        String visitorCode = ((EditText) findViewById(R.id.
            inputVisitorCode)).getText().toString();
        if (validateVisitorCode(visitorCode)) {
            // Successfully logged in as visitor
            navigateToReservationScreen();
        } else {
            // Show error message
            Toast.makeText(this, "Neplatny kod", Toast.LENGTH_SHORT).
                show();
        }
    }

    private boolean validateVisitorCode(String code) {
        // Validation logic for visitor code
        return !code.isEmpty(); // Placeholder for validation logic
    }

    private void navigateToReservationScreen() {
        // Navigate to the screen where the user can reserve parking
            spots
        Intent intent = new Intent(MainActivity.this,
            ReservationActivity.class);
        startActivity(intent);
    }
}

```

Listing 14: MainActivity.java

Nasledoval Use Case 2 pre rezerváciu parkovacích miest s triedou ReservationActivity.java.

2. Rezervácia parkovacieho miesta (Use Case 2)

Popis:

- Užívateľ si rezervuje parkovacie miesto po prihlásení.

Vylepšenie:

- **Výber parkovacieho miesta:** Užívateľ si môže vybrať parkovacie miesto zo zoznamu dostupných miest.
- **Zobrazenie dostupných miest:** Aplikácia by mala načítať a zobraziť dostupné parkovacie miesta zo servera alebo databázy.
- **Potvrdenie rezervácie:** Po výbere miesta a časového intervalu užívateľ potvrdí rezerváciu.

Obr. 16: Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 2

```
public class ReservationActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private ParkingSpotAdapter adapter;
    private List<ParkingSpot> parkingSpots;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation);

        recyclerView = findViewById(R.id.recyclerView);
        parkingSpots = new ArrayList<>();

        // Populate parking spots (could be from a database or server)
        parkingSpots.add(new ParkingSpot("A1", "A1", true, "student"));
        parkingSpots.add(new ParkingSpot("B2", "B2", false, "employee"));
        // Add more parking spots...

        adapter = new ParkingSpotAdapter(parkingSpots, spot -> reserveSpot(
            spot));
        recyclerView.setAdapter(adapter);
    }

    private void reserveSpot(ParkingSpot spot) {
        // Handle reservation process
        if (spot.isAvailable()) {
            // Make reservation
            Toast.makeText(this, "Rezervacia uspesna!", Toast.LENGTH_SHORT)
                .show();
            spot.setAvailable(false); // Mark as reserved
            adapter.notifyDataSetChanged();
        } else {
            Toast.makeText(this, "Miesto uz je obsadene", Toast.
                LENGTH_SHORT).show();
        }
    }
}
```

Listing 15: ReservationActivity.java

Nasledoval Use Case 3 pre zrušenie rezervácie s triedou ReservationListActivity.java.

3. Zrušenie rezervácie (Use Case 3)

Popis:

- Užívateľia môžu zrušiť svoju rezerváciu, ak je to ešte možné (do 30 minút pred začiatkom).

Vylepšenie:

- **Zobrazenie rezervácií:** Užívateľ vidí svoje aktívne rezervácie.
- **Zrušenie rezervácie:** Ak je zrušenie povolené (do 30 minút), rezervácia bude zrušená.

Obr. 17: Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 3

```
public class ReservationListActivity extends AppCompatActivity {

    private ListView listView;
    private List<Reservation> reservations;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation_list);

        listView = findViewById(R.id.listView);
        reservations = getReservationsFromDb(); // Fetch reservations from DB

        ArrayAdapter<Reservation> adapter = new ArrayAdapter<>(this,
            android.R.layout.simple_list_item_1, reservations);
        listView.setAdapter(adapter);

        listView.setOnItemClickListener((parent, view, position, id) -> {
            Reservation reservation = reservations.get(position);
            showCancelDialog(reservation);
        });
    }

    private void showCancelDialog(Reservation reservation) {
        new AlertDialog.Builder(this)
            .setMessage("Chcete zrušiť túto rezerváciu?")
            .setPositiveButton("Zrušiť", (dialog, which) ->
                cancelReservation(reservation))
            .setNegativeButton("Zrušiť", null)
            .show();
    }

    private void cancelReservation(Reservation reservation) {
        if (reservation.canBeCancelled()) {
            // Cancel reservation
            Toast.makeText(this, "Rezervácia zrušená!", Toast.LENGTH_SHORT)
                .show();
        } else {
            Toast.makeText(this, "Zrušenie nie je možné, rezervácia už
                začala.", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Listing 16: ReservationListActivity.java

Nasledoval Use Case 4 pre zmenu rezervácie s aktualizovanou triedou ReservationListActivity.java.

4. Zmena rezervácie (Use Case 4)

Popis:

- Užívatelia môžu meniť čas alebo miesto svojej rezervácie.

Vylepšenie:

- **Úprava rezervácie:** Užívatelia si môžu vybrať novú časovú alebo miestnu hodnotu.

Obr. 18: Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 4

```
private void showChangeDialog(Reservation reservation) {
    new AlertDialog.Builder(this)
        .setMessage("Chcete zmeniť tuto rezerváciu?")
        .setPositiveButton("Zmeniť", (dialog, which) ->
            changeReservation(reservation))
        .setNegativeButton("Zrusiť", null)
        .show();
}

private void changeReservation(Reservation reservation) {
    // Open a new screen or dialog to modify the reservation
    // (either time or parking spot)
}
```

Listing 17: ReservationListActivity.java - extended

Nasledoval Use Case 5 pre náhľad na dostupné parkovacie miesta a Use Case 6 pre odhlásenie z aplikácie s triedou SettingsActivity.java.

5. Náhľad na dostupné parkovacie miesta (Use Case 5)

Popis:

- Užívatelia môžu zobrazit' dostupné parkovacie miesta na mape.

Vylepšenie:

- **Zobrazenie mapy:** Implementovať mapu s dostupnými parkovacími miestami.

6. Odhlásenie z aplikácie (Use Case 6)

Popis:

- Užívatelia sa môžu odhlásiť zo systému.

Vylepšenie:

- **Odhlásenie:** Pridať možnosť odhlásenia z aplikácie v nastaveniach.

Obr. 19: Odpoveď ChatGPT na príkaz s popismi Use Cases - Use Case 5 a 6

```
public class SettingsActivity extends AppCompatActivity {

    private Button btnLogout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        btnLogout = findViewById(R.id.btnLogout);
        btnLogout.setOnClickListener(v -> logout());
    }

    private void logout() {
        // Handle logout
        Intent intent = new Intent(SettingsActivity.this,
            MainActivity.class);
        startActivity(intent);
        finish();
    }
}
```

Listing 18: SettingsActivity.java

Následne sme ChatGPT požiadali aj o aktualizáciu XML súborov.

1. activity_login.xml (Prihlasovacia obrazovka)

Táto obrazovka obsahuje dve tlačidlá: jedno pre prihlásenie cez SSO a druhé pre prihlásenie s jednorazovým kódom.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/loginTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Prihlasenie do aplikacie"
        android:textSize="24sp"
        android:layout_marginBottom="32dp" />

    <!-- Button for SSO login -->
    <Button
        android:id="@+id/btnSSOLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prihlasit sa cez SSO"
        android:layout_marginBottom="16dp" />

    <!-- EditText for visitor one-time code -->
    <EditText
        android:id="@+id/inputVisitorCode"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Jednorazovy kod"
        android:inputType="text"
        android:layout_marginBottom="16dp" />

    <!-- Button for visitor login -->
    <Button
        android:id="@+id/btnVisitorLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prihlasit sa jednorazovym kodom" />

</LinearLayout>
```

Listing 19: activity_login.xml

2. activity_reservation.xml (Obrazovka pre rezerváciu parkovacieho miesta)

Táto obrazovka zobrazuje dostupné parkovacie miesta v RecyclerView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/reservationTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Rezervacia parkovacieho miesta"
        android:textSize="24sp"
        android:layout_marginBottom="16dp" />

    <!-- RecyclerView to show parking spots -->
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Listing 20: activity_reservation.xml

3. activity_reservation_list.xml (Obrazovka pre zobrazenie rezervácií)

Táto obrazovka zobrazuje zoznam existujúcich rezervácií užívateľa v ListView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/reservationListTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Moje rezervacie"
        android:textSize="24sp"
        android:layout_marginBottom="16dp" />

    <!-- ListView to display user's reservations -->
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Listing 21: activity_reservation_list.xml

4. activity_settings.xml (Obrazovka pre nastavenia a odhlásenie)

Táto obrazovka obsahuje tlačidlo pre odhlásenie užívateľa.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/settingsTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nastavenia"
        android:textSize="24sp"
        android:layout_marginBottom="32dp" />

    <!-- Button for logout -->
    <Button
        android:id="@+id/btnLogout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Odhlásiť sa" />
</LinearLayout>
```

Listing 22: activity_settings.xml

5. activity_parking_map.xml (Obrazovka pre náhľad dostupných parkovacích miest na mape)

Táto obrazovka zobrazuje mapu parkoviska a dostupné parkovacie miesta.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Map fragment for displaying parking spots -->
    <com.google.android.gms.maps.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

Listing 23: activity_parking_map.xml

6. item_parking_spot.xml (Layout pre parkovacie miesto v RecyclerView)

Tento layout sa použije pre jednotlivé parkovacie miesta v RecyclerView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="16dp"
    android:layout_marginBottom="8dp"
    android:background="?android:attr/selectableItemBackground">

    <TextView
        android:id="@+id/spotName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Miesto A1"
        android:textSize="18sp"
        android:layout_marginEnd="16dp" />

    <TextView
        android:id="@+id/spotStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Volne"
        android:textSize="18sp"
        android:textColor="@android:color/black" />

</LinearLayout>
```

Listing 24: item_parking_spot.xml

7. dialog_cancel_reservation.xml (Dialog pre zrušenie rezervácie)

Tento dialog sa zobrazí, keď užívateľ zruší svoju rezerváciu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/cancelReservationMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Chcete zrušiť túto rezerváciu?"
        android:textSize="18sp"
        android:layout_marginBottom="16dp" />

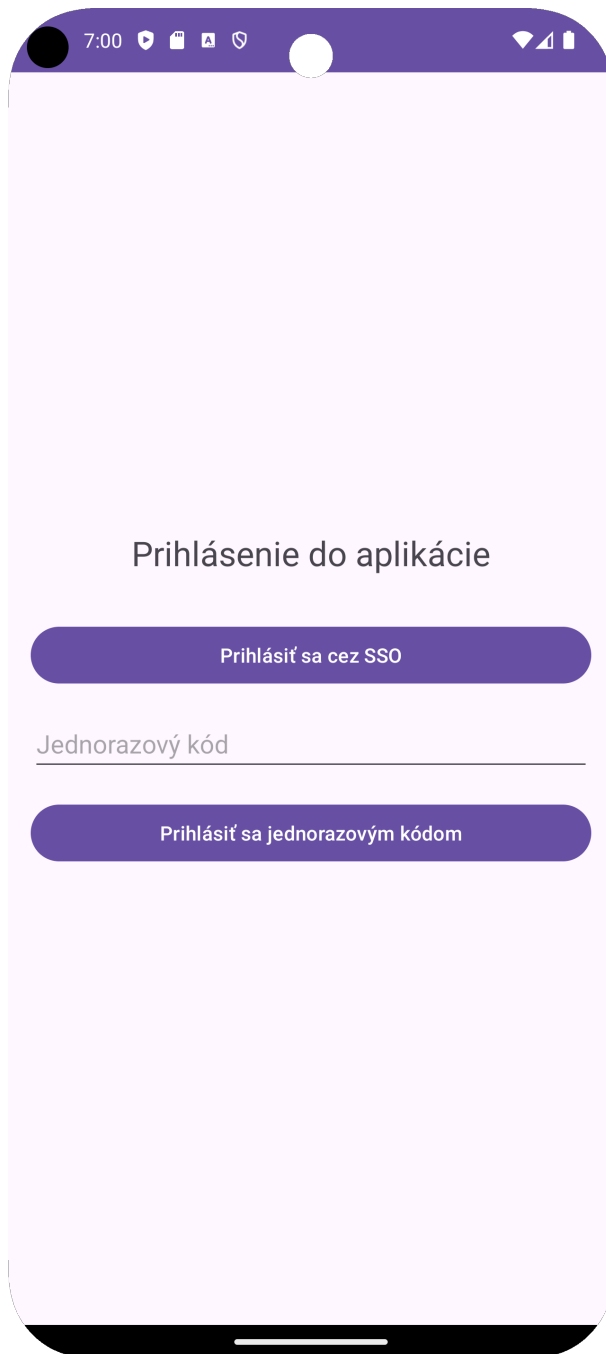
    <Button
        android:id="@+id/btnCancelReservation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Zrušiť rezerváciu"
        android:textColor="@android:color/white"
        android:background="@color/colorAccent" />

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Zrušiť"
        android:textColor="@android:color/black"
        android:layout_marginTop="8dp" />
</LinearLayout>
```

Listing 25: dialog_cancel_reservation.xml

2.2 Pridanie Use Case scenárov

Po úspešnom pridaní kódu z kapitoly 2.1 do Android Studio sme dostali už niečo, povedzme, klikateľné v podobe login screenu aplikácie. Samozrejme, po kliknutí na tlačidlá sa nestane vôbec nič, nakoľko stále nemáme implementované chovanie niektorých metód.



Obr. 20: Stav aplikácie po pridaní popisu Use Cases

Následne sme predali ChatGPT Use Cases scenáre. Z nejakého dôvodu nám upravil len XML súbory.

Pridávam scenárov Use Caseov. Dá sa na základe nich niečo upraviť v kóde?

Scenár č.1: Prihlásenie užívateľa

Tento scenár opisuje proces prihlásenia do systému na správu parkovacích miest. Používateľ sa môže prihlásiť buď cez Single Sign-On (SSO), ak je študentom alebo zamestnancom VŠB-TUO, alebo pomocou jednorazového kódu, ak ide o návštevníka.

Kroky:

1. Spustenie aplikácie
 - Používateľ otvorí systém na správu parkovania.
2. Výber spôsobu prihlásenia
 - Systém zobrazí možnosti autentifikácie:
 - Prihlásiť sa cez SSO
 - Prihlásiť sa jednorazovým kódom (návštevníci)
3. Prihlásenie cez SSO (študenti a zamestnanci)
 - Ak si používateľ zvolí SSO, systém ho presmeruje na autentifikačný server univerzity.
 - Používateľ zadá svoje univerzitné prihlasovacie údaje, ktoré sú následne overené.
 - Po úspešnom overení je používateľ automaticky prihlásený do systému.
4. Prihlásenie cez jednorazový kód (návštevníci)
 - Ak si používateľ zvolí jednorazový kód, systém vyžiada e-mailovú adresu, na ktorú odošle kód.

Obr. 21: Príkaz s popismi scenárov Use Cases

Na základe pridaných scenárov Use Case môžeme upraviť kód a XML súbory aplikácie, aby lepšie vyhovovali požiadavkám uvedeným v týchto scenároch. Tu sú niektoré zmeny a vylepšenia, ktoré môžeme implementovať do aplikácie.

Zmeny v kóde:

1. Prihlásenie užívateľa

- **Prepojenie s autentifikačným serverom:** Kód na prihlásenie cez **SSO** alebo **jednorazový kód** bude treba prispôsobiť tak, aby zvládol oba scenáre.
 - Pri prihlásení cez SSO sa systém presmeruje na autentifikačný server univerzity, kde sa vykoná autentifikácia.
 - Pri prihlásení cez jednorazový kód sa najprv vyžiada e-mailová adresa a následne sa overí platnosť kódu.
- **Chybové hlásenia:** Ak prihlásenie zlyhá (nesprávny kód alebo údaje), aplikácia musí zobraziť chybové hlásenia, ktoré informujú používateľa o probléme a umožnia mu opraviť chybu.

2. Náhľad na parkoviská a rezervácia parkovacieho miesta

- **Zobrazenie dostupných miest na mape:** V aplikácii bude potrebné implementovať interaktívnu mapu (pomocou Google Maps alebo podobnej knižnice), ktorá bude zobrazovať aktuálny stav parkovísk a ich dostupnosť.
- **Výber parkovacieho miesta:** Používateľ si bude môcť vybrať parkovisko a parkovacie miesto, ktoré je k dispozícii.
- **Overenie dostupnosti miesta pred rezerváciou:** Pred potvrdením rezervácie bude systém overovať, či je miesto stále voľné.
- **Zrušenie rezervácie:** Systém musí podporovať zrušenie rezervácie, ale len v rámci stanovených podmienok (do 30 minút pred začiatkom). Ak sa používateľ pokúsi zrušiť rezerváciu po tejto lehote, systém mu to nedovolí a zobrazí varovanie.

3. Scenáre alternatívnych situácií:

- **Nedostupnosť parkovacieho miesta:** Ak už nie je dostupné parkovacie miesto, aplikácia by mala ponúknuť alternatívy, ako nastaviť notifikáciu o uvoľnení miesta alebo výber iného času/miesta.
- **Rezervácia nie je možná:** Ak sa používateľ pokúsi rezervovať miesto, ktoré je obsadené alebo pre jeho rolu nedostupné, aplikácia by mala zobraziť upozornenie a ponúknuť alternatívu.

Obr. 22: Odpoveď ChatGPT na príkaz s popismi scenárov Use Cases

1. activity_login.xml

Aktualizujeme obrazovku prihlásenia, aby bolo jasné, že používateľ môže zadať jednorazový kód alebo vybrať SSO možnosť. Budeme mať aj upozornenia pre chybné prihlásenie.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/loginTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Prihlásenie do aplikácie"
        android:textSize="24sp"
        android:layout_marginBottom="32dp" />

    <!-- Button for SSO login -->
    <Button
        android:id="@+id/btnSSOLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prihlasiť sa cez SSO"
        android:layout_marginBottom="16dp" />

    <!-- EditText for visitor one-time code -->
    <EditText
        android:id="@+id/inputVisitorCode"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Jednorazový kód"
        android:inputType="text"
        android:layout_marginBottom="16dp"
        android:visibility="gone" />

    <!-- Button for visitor login -->
    <Button
        android:id="@+id/btnVisitorLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prihlasiť sa jednorazovým kódom"
        android:layout_marginBottom="16dp" />

    <!-- TextView to show error messages if login fails -->
    <TextView
        android:id="@+id/loginError"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Chyba pri prihlásení. Skúste to znova."
        android:textColor="@android:color/holo_red_dark"
        android:visibility="gone"
        android:layout_marginTop="16dp"/>
</LinearLayout>
```

Listing 26: activity_login.xml

2. activity_parking_map.xml

Pridáme interaktívnu mapu pre zobrazenie dostupných parkovacích miest.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Google Map to show available parking spots -->
    <com.google.android.gms.maps.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- Button to select parking spot -->
    <Button
        android:id="@+id/btnSelectSpot"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Rezervovat"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />
</FrameLayout>
```

Listing 27: activity_parking_map.xml

3. activity_reservation.xml

Zobrazíme dostupné parkovacie miesta v RecyclerView alebo ListView (podľa potreby).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/reservationTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Rezervacia parkovacieho miesta"
        android:textSize="24sp"
        android:layout_marginBottom="16dp" />

    <!-- RecyclerView to show parking spots -->
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Listing 28: activity_reservation.xml

4. Zrušenie rezervácie

Pridáme Dialog na zrušenie rezervácie, ktorý zohľadňuje podmienky (napr. nie menej ako 30 minút pred začiatkom).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/cancelReservationMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Chcete zrusit tuto rezervaciju?"
        android:textSize="18sp"
        android:layout_marginBottom="16dp" />

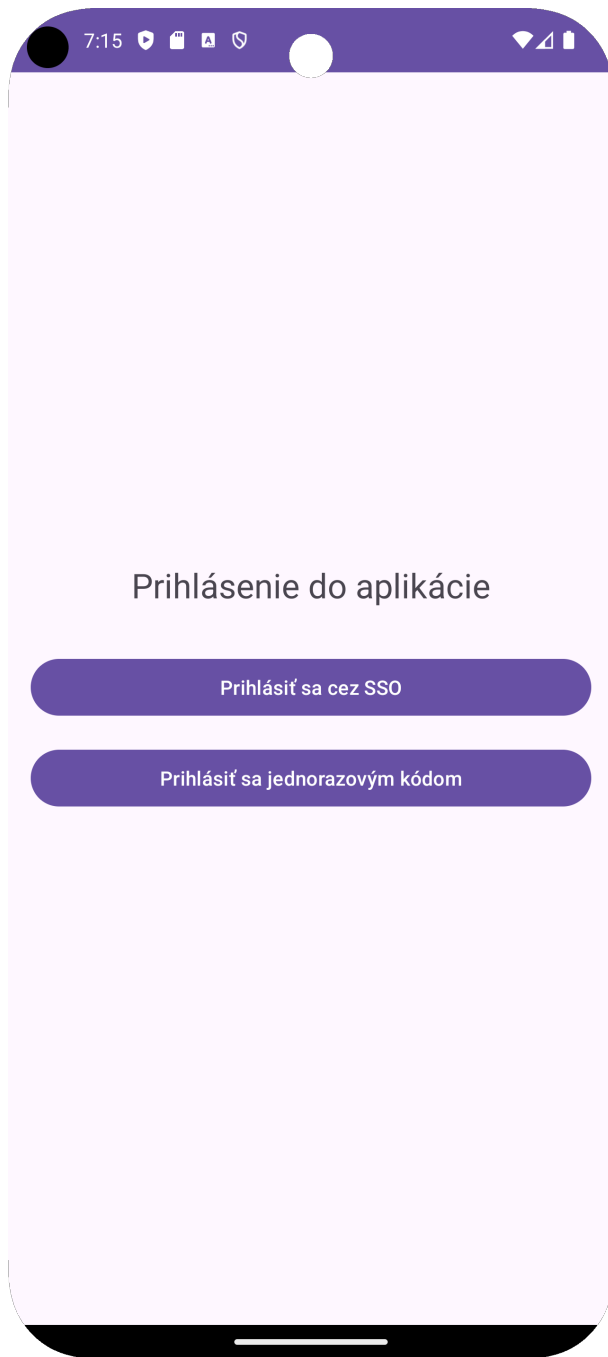
    <Button
        android:id="@+id/btnCancelReservation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Zrusit rezervaciju"
        android:textColor="@android:color/white"
        android:background="@color/colorAccent" />

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Zrusit"
        android:textColor="@android:color/black"
        android:layout_marginTop="8dp" />
</LinearLayout>
```

Listing 29: activity_reservation.xml

2.3 Výsledok

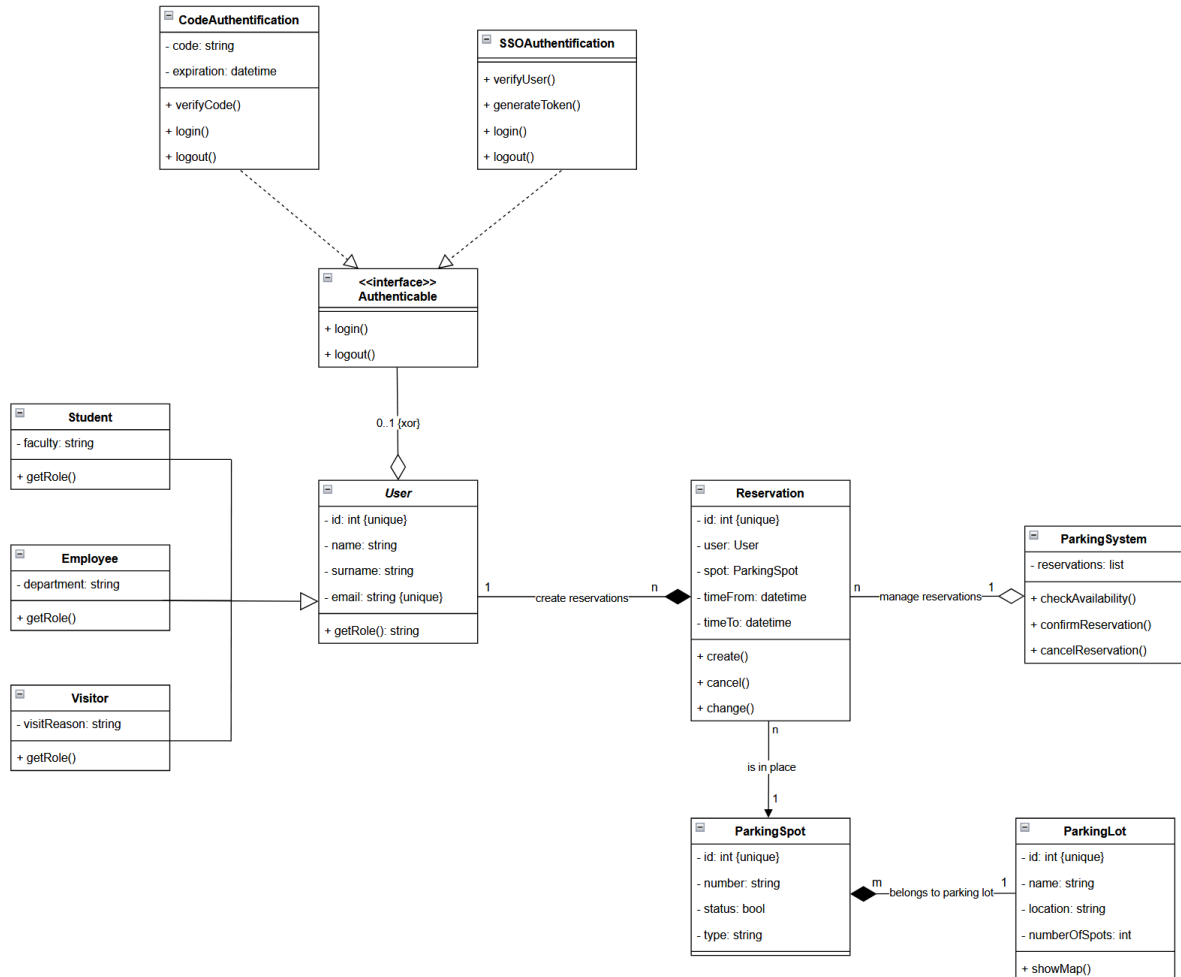
Po kapitole 3 môžeme povedať, že máme základné kódy chovania aplikácie, avšak väčšinou len ich deklarácie. Po zapnutí aplikácie sa zobrazí úvodná obrazovka s login screenom, kde je možné kliknúť na 2 tlačítka, avšak nič sa nestane.



Obr. 23: Stav aplikácie po druhej kapitole

3 Class diagram

Obsahom tejto kapitoly je začleniť do našej aplikácie triedne diagramy. Z nášho triedneho diagramu sme vytvorili PlantUML kód, ktorý dokáže ChatGPT lepšie spracovať.



Obr. 24: Class diagram

3.1 Class, class, interface...

```
@startuml

interface Authenticable {
    +login()
    +logout()
}

abstract class User {
    -id: int {unique}
    -name: string
    -surname: string
    -email: string {unique}
    +getRole(): string
}

class CodeAuthentication {
    -code: string
    -expiration: datetime
    +verifyCode()
    +login()
    +logout()
}

class SSOAuthentication {
    +verifyUser()
    +generateToken()
    +login()
    +logout()
}

CodeAuthentication ..|> Authenticable
SSOAuthentication ..|> Authenticable
User ..|> Authenticable

class Student {
    -faculty: string
    +getRole(): string
}

class Employee {
    -department: string
    +getRole(): string
}

class Visitor {
    -visitReason: string
    +getRole(): string
}
```

```

}

User <|-- Employee
User <|-- Student
User <|-- Visitor

class ParkingSystem {
    -reservations: list
    +checkAvailability()
    +confirmReservation()
    +cancelReservation()
}

class Reservation {
    -id: int {unique}
    -user: User
    -spot: ParkingSpot
    -timeFrom: datetime
    -timeTo: datetime
    +create()
    +cancel()
    +change()
}

class ParkingSpot {
    -id: int {unique}
    -number: string
    -status: bool
    -type: string
}

class ParkingLot {
    -id: int {unique}
    -name: string
    -location: string
    -numberOfSpots: int
    +showMap()
}

User "1" -- "n" Reservation : create reservations
ParkingSystem "1" -- "n" Reservation : manage reservations
Reservation "n" -- "1" ParkingSpot : is in place
ParkingSpot "m" -- "1" ParkingLot : belongs to parking lot

@enduml

```

Listing 30: PlantUML reprezentácia Class diagramu

Tu je PlantUML kód, ktorý znázorňuje class diagram mojej aplikácie.
Dokážeš na jeho základe upraviť moju aplikáciu?

@startuml

```
interface Authenticable {  
    +login()  
    +logout()  
}
```

```
abstract class User {  
    -id: int {unique}  
    -name: string  
    -surname: string  
    -email: string {unique}  
    +getRole(): string  
}
```

```
class CodeAuthentication {  
    -code: string  
    -expiration: datetime
```

Obr. 25: Príkaz s PlantUML reprezentáciou Class diagramu

Na základe poskytnutého PlantUML kódu môžeme prispôbiť tvoju aplikáciu tak, aby reflektovala danú architektúru objektov. Tento diagram vyjadruje štruktúru tried, ktoré sú zodpovedné za správu používateľov, autentifikáciu a parkovanie. Tu je niekoľko spôsobov, ako by sa tieto zmeny mohli implementovať do tvojej aplikácie.

Zmeny v kóde na základe diagramu:

1. Implementácia rozhrania `Authenticable` a autentifikačných mechanizmov (SSO a kód)

- **Rozhranie `Authenticable`** definuje metódy pre prihlásenie a odhlásenie, ktoré musí implementovať každá trieda autentifikácie. V aplikácii budeš mať dve autentifikačné stratégie:
 - `SSOAuthentication` : Implementuje autentifikáciu cez Single Sign-On (SSO), overenie používateľa a generovanie tokenu.
 - `CodeAuthentication` : Implementuje autentifikáciu cez jednorazový kód.

2. Dedičnosť používateľských tried

- Existujú tri typy používateľov, ktoré dedí trieda `User` :
 - `Student` : Pre študentov, ktorí majú fakultu.
 - `Employee` : Pre zamestnancov, ktorí majú priradené oddelenie.
 - `Visitor` : Pre návštevníkov, ktorí majú dôvod návštevy.
- Všetky tieto triedy implementujú metódu `getRole()` , ktorá vracia rolu používateľa.

3. Rezervácia parkovacích miest a správa parkovacích miest

- `ParkingSystem` spravuje rezervácie parkovacích miest, pričom ponúka metódy na kontrolu dostupnosti, potvrdenie rezervácie a zrušenie rezervácie.
- `Reservation` reprezentuje jednotlivé rezervácie, ktoré sú priradené k používateľom a parkovacím miestam. Každá rezervácia obsahuje časové údaje `timeFrom` a `timeTo` .
- `ParkingSpot` reprezentuje konkrétne parkovacie miesto s atribútmi ako `status` (voľné/obsadené) a `type` (typ miesta, napríklad pre študentov, zamestnancov alebo návštevníkov).
- `ParkingLot` obsahuje zoznam parkovacích miest a umožňuje zobrazenie mapy.

Obr. 26: Odpoveď ChatGPT na príkaz s PlantUML reprezentáciou Class diagramu

Implementácia tried a metód v aplikácii:

1. Authenticable rozhranie

```
public interface Authenticable {  
    void login();  
    void logout();  
}
```

Listing 31: Authenticable.java

2. Triedy pre autentifikáciu

- SSOAuthentication:

```
public class SSOAuthentication implements Authenticable {  
    private String token;  
  
    @Override  
    public void login() {  
        // Presmerovanie na SSO server a získanie tokenu  
        verifyUser();  
        generateToken();  
    }  
  
    @Override  
    public void logout() {  
        token = null;  
    }  
  
    public void verifyUser() {  
        // Overenie pouzivatela cez SSO  
    }  
  
    public void generateToken() {  
        // Generovanie tokenu  
    }  
}
```

Listing 32: SSOAuthentication.java

- CodeAuthentication:

```
public class CodeAuthentication implements Authenticable {
    private String code;
    private LocalDateTime expiration;

    @Override
    public void login() {
        verifyCode();
    }

    @Override
    public void logout() {
        code = null;
    }

    public void verifyCode() {
        // Overenie jednorazoveho kodu
    }
}
```

Listing 33: CodeAuthentication.java

3. Triedy pre používateľov (User, Student, Employee, Visitor)

```
public abstract class User implements Authenticable {
    private int id;
    private String name;
    private String surname;
    private String email;

    public abstract String getRole();
}

public class Student extends User {
    private String faculty;

    @Override
    public String getRole() {
        return "Student";
    }
}

public class Employee extends User {
    private String department;

    @Override
    public String getRole() {
        return "Employee";
    }
}

public class Visitor extends User {
    private String visitReason;

    @Override
    public String getRole() {
        return "Visitor";
    }
}
```

Listing 34: User.java

4. Trieda pre parkovací systém a rezervácie

- ParkingSystem:

```
import java.util.List;

public class ParkingSystem {
    private List<Reservation> reservations;

    public boolean checkAvailability(ParkingSpot spot) {
        // Skontrolovanie dostupnosti parkovacieho miesta
        return spot.isAvailable();
    }

    public void confirmReservation(Reservation reservation) {
        // Potvrdenie rezervacie
        reservations.add(reservation);
    }

    public void cancelReservation(Reservation reservation) {
        // Zrusenie rezervacie
        reservations.remove(reservation);
    }
}
```

Listing 35: ParkingSystem.java

- Reservation:

```
import java.time.LocalDateTime;

public class Reservation {
    private int id;
    private User user;
    private ParkingSpot spot;
    private LocalDateTime timeFrom;
    private LocalDateTime timeTo;

    public void create() {
        // Vytvorenie rezervacie
    }

    public void cancel() {
        // Zrusenie rezervacie
    }

    public void change() {
        // Zmena rezervacie
    }
}
```

Listing 36: Reservation.java

- ParkingSpot:

```
public class ParkingSpot {  
    private int id;  
    private String number;  
    private boolean status; // true = volne, false = obsadene  
    private String type;  
  
    public boolean isAvailable() {  
        return status;  
    }  
}
```

Listing 37: ParkingSpot.java

- ParkingLot:

```
import java.util.List;  
  
public class ParkingLot {  
    private int id;  
    private String name;  
    private String location;  
    private int numberOfSpots;  
    private List<ParkingSpot> parkingSpots;  
  
    public void showMap() {  
        // Zobrazit mapu parkoviska  
    }  
}
```

Listing 38: ParkingLot.java

3.2 Ukladáme dáta

Ako sme spomínali v kapitole 1.3, ChatGPT nám ponúkol možnosť využitia Room databázy v našej aplikácii. Využili sme preto túto možnosť, a nechali sme si vytvoriť kód, ktorý by nám umožnil používať takúto funkcionality.

Vedel by si tieto kódy spojiť s celým celkom tak, aby sa aplikácia dala spustiť a dala sa v nej robiť konkrétne činnosti - aj s použitím vyššie použitej room databázy?

Obr. 27: Príkaz pre začatie používania Room databázy

Áno, môžem ti pomôcť so spojením týchto kódov a pridať podporu pre **Room Database** v aplikácii. Room je knižnica pre SQLite databázu v Android aplikáciách, ktorá umožňuje ukladanie dát v relačnej databáze a poskytuje abstrahovanú API pre interakciu s databázou.

Tu je ako by mohol vyzeráť celý kód pre Android aplikáciu, ktorá kombinuje tvoje požiadavky a používa **Room Database** na správu používateľov, rezervácií a parkovacích miest. Tento príklad ukáže základnú implementáciu s používaním tvojich tried a zahrnutím Room databázy.

Obr. 28: Odpoveď ChatGPT na príkaz pre začatie používania Room databázy

Implementáciu sme dostali rozdelenú na viacero krokov.

Krok 1: Nastavenie Room Database

Prvým krokom je nastaviť základy Room databázy v aplikácii. Pre tento účel je potrebné definovať **entity** (tabuľky), **DAO** (Data Access Objects) a **Database** triedu.

Obr. 29: Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 1.krok

1.1. Entita pre používateľov

```
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "users")
public abstract class User {
    @PrimaryKey(autoGenerate = true)
    private int id;
    private String name;
    private String surname;
    private String email;

    public abstract String getRole();

    // Getters and Setters
}
```

Listing 39: User.java

1.2. Entity pre rezervácie

```
import androidx.room.Entity;
import androidx.room.ForeignKey;
import androidx.room.PrimaryKey;

import java.time.LocalDateTime;

@Entity(tableName = "reservations", foreignKeys = {
    @ForeignKey(entity = User.class, parentColumns = "id",
        childColumns = "userId"),
    @ForeignKey(entity = ParkingSpot.class, parentColumns = "id",
        childColumns = "spotId")
})
public class Reservation {
    @PrimaryKey(autoGenerate = true)
    private int id;
    private int userId;
    private int spotId;
    private LocalDateTime timeFrom;
    private LocalDateTime timeTo;

    // Getters and Setters
}
```

Listing 40: Reservation.java

1.3. Entita pre parkovacie miesta

```
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "parking_spots")
public class ParkingSpot {
    @PrimaryKey(autoGenerate = true)
    private int id;
    private String number;
    private boolean status; // true = volne, false = obsadene
    private String type;

    // Getters and Setters
}
```

Listing 41: ParkingSpot.java

1.4. Entita pre parkoviská

```
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "parking_lots")
public class ParkingLot {
    @PrimaryKey(autoGenerate = true)
    private int id;
    private String name;
    private String location;
    private int numberOfSpots;

    // Getters and Setters
}
```

Listing 42: ParkingLot.java

Krok 2: DAO (Data Access Object)

DAO je zodpovedné za interakciu s databázou, teda operácie ako vloženie, čítanie, aktualizovanie a mazanie záznamov.

Obr. 30: Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 2.krok

2.1. DAO pre používateľov

```
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

@Dao
public interface UserDao {
    @Insert
    void insertUser(User user);

    @Query("SELECT * FROM users")
    List<User> getAllUsers();

    @Query("SELECT * FROM users WHERE email = :email LIMIT 1")
    User getUserByEmail(String email);
}
```

Listing 43: UserDAO.java

2.2. DAO pre rezervácie

```
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

@Dao
public interface ReservationDao {
    @Insert
    void insertReservation(Reservation reservation);

    @Query("SELECT * FROM reservations WHERE userId = :userId")
    List<Reservation> getReservationsByUserId(int userId);

    @Query("DELETE FROM reservations WHERE id = :reservationId")
    void deleteReservation(int reservationId);
}
```

Listing 44: ReservationDAO.java

2.3. DAO pre parkovacie miesta

```
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

@Dao
public interface ParkingSpotDao {
    @Insert
    void insertParkingSpot(ParkingSpot parkingSpot);

    @Query("SELECT * FROM parking_spots WHERE id = :spotId LIMIT 1")
    ParkingSpot getParkingSpotById(int spotId);
}
```

Listing 45: ParkingSpotDao.java

Krok 3: Room Database

Trieda **RoomDatabase** bude obsahovať metódy na získanie DAO pre prístup k databáze.

Obr. 31: Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 3.krok

```
import androidx.room.Database;
import androidx.room.RoomDatabase;

@Database(entities = {User.class, Reservation.class, ParkingSpot.class, ParkingLot.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
    public abstract ReservationDao reservationDao();
    public abstract ParkingSpotDao parkingSpotDao();
}
```

Listing 46: AppDatabase.java

Krok 4: Príklad implementácie aktivít a logiky

Teraz vytvoríme implementáciu pre autentifikáciu a správu rezervácií. Tento kód ukazuje, ako môžeš používať databázu v aktivite.

Obr. 32: Odpoveď ChatGPT na príkaz pre začatie používania Room databázy - 4.krok

4.1. Autentifikácia cez SSO alebo jednorazový kód

```
public class AuthActivity extends AppCompatActivity {
    private AppDatabase db;
    private UserDao userDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_auth);

        db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        userDao = db.userDao();

        // Tu budeš implementovať prihlásenie pomocou SSO alebo kodu
        authenticateUser();
    }

    private void authenticateUser() {
        // Skontroluj prihlasovacie údaje a naitaj používateľa z
        // databázy
        String email = "user@example.com"; // Příklad emailu
        User user = userDao.getUserByEmail(email);
        if (user != null) {
            // Pokračuj v aplikácii ako prihlásený používateľ
        }
    }
}
```

Listing 47: AuthActivity.java

4.2. Rezervácia parkovacieho miesta

```
public class ReservationActivity extends AppCompatActivity {
    private AppDatabase db;
    private ReservationDao reservationDao;
    private ParkingSpotDao parkingSpotDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation);

        db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        reservationDao = db.reservationDao();
        parkingSpotDao = db.parkingSpotDao();

        // Rezervacia parkovacieho miesta
        reserveParkingSpot(1, 1); // Predstavme si, ze pouzivatel a
            miesto su uz zname
    }

    private void reserveParkingSpot(int userId, int spotId) {
        ParkingSpot spot = parkingSpotDao.getParkingSpotById(spotId);
        if (spot != null && spot.isAvailable()) {
            Reservation reservation = new Reservation();
            reservation.setUserId(userId);
            reservation.setSpotId(spotId);
            reservation.setTimeFrom(LocalDateTime.now());
            reservation.setTimeTo(LocalDateTime.now().plusHours(1));

            reservationDao.insertReservation(reservation);
        }
    }
}
```

Listing 48: ReservationActivity.java

3.3 Ono to nesype

Po prekopírovaní zdrojových kódov z kapitoly 3 do Android Studio sme zistili, že po nahraní Class diagramu ChatGPT úplne odignoroval všetok predom generovaný kód, čiže dané triedy vôbec nenaväzujú na nič. Taktiež spolu s používaním Room databázy sme dostali asi bambilión errorov a warningov, čo robilo náš projekt prakticky nespustiteľným.

Preto sme požiadali ChatGPT o to, aby porovnal dve verzie zdrojových kódov (pred a po vložení Class diagramu) a vytvoril z neho funkčnú a spustiteľnú verziu oboch kódov.

V súvislosti s vyššie spomínaným Class diagramom, a zároveň s úmyslom pripojiť aplikáciu k Room databázi pre ukladanie údajov, zoberie tieto dva zdrojové kódy a spoj ich do jedného tak, aby dávali zmysel.

```
@Database(entities = {Reservation.class}, version = 1)
@TypeConverters({Converters.class})
public abstract class AppDatabase extends RoomDatabase {
    public abstract ReservationDao reservationDao();

    private static AppDatabase instance;

    public static synchronized AppDatabase getInstance(Context
context) {
        if (instance == null) {
            instance =
Room.databaseBuilder(context.getApplicationContext(),
                        AppDatabase.class, "parking_database")
                        .fallbackToDestructiveMigration()
                        .build();
        }
        return instance;
    }
}

@Database(entities = {User.class, Reservation.class, ParkingSpot.class,
ParkingLot.class}, version = 1)
```

Obr. 33: Príkaz pre zlúčenie dvoch kódov do jedného

Jasné! Zdá sa, že máš **dva zdrojové kódy**, ktoré definujú dve rôzne verzie triedy `AppDatabase`. V Android Room databáze však môže existovať len **jedna centrálna trieda databázy**, ktorá obsahuje všetky entity a DAO.

Tu je **spojená verzia** tvojej `AppDatabase`, ktorá kombinuje všetky entity (`User`, `Reservation`, `ParkingSpot`, `ParkingLot`) aj DAO rozhrania (`UserDao`, `ReservationDao`, `ParkingSpotDao`) a zároveň si zachováva aj `TypeConverter`, ak pracuješ s typmi ako `LocalDateTime`.

Obr. 34: Odpoveď ChatGPT na príkaz pre zlúčenie dvoch kódov do jedného

```
import android.content.Context;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import androidx.room.TypeConverters;

// Pridaj všetky potrebné entity a DAO
@Database(
    entities = {
        User.class,
        Reservation.class,
        ParkingSpot.class,
        CodeAuthentication.class // Pridane CodeAuthentication ako entitu
    },
    version = 1
)
@TypeConverters({Converters.class})
public abstract class AppDatabase extends RoomDatabase {

    public abstract UserDao userDao();
    public abstract ReservationDao reservationDao();
    public abstract ParkingSpotDao parkingSpotDao();
    public abstract CodeDao codeDao(); // Pridanie DAO pre CodeAuthentication

    private static volatile AppDatabase instance;

    public static synchronized AppDatabase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(
                context.getApplicationContext(),
                AppDatabase.class,
                "parking_database"
            )
                .fallbackToDestructiveMigration()
                .build();
        }
        return instance;
    }
}
```

Listing 49: AppDatabase.java

Rovnakým spôsobom sme postupovali pre všetky zdrojové kódy v aplikácii. Nižšie pre skrátenie dokumentácie uvádzame len výsledné zlúčeniny kódov.

V prípade User.java nám bolo odporúčané, že Room priamo nepodporuje dedičnosť entít, a že ak chceme použiť Visitor, Student, Employee ako špecializované verzie User, tak máme použiť jednu entitu User, ktorá má pole role ("student", "visitor", "employee"). Tým pádom nám zaniká potreba mať tieto triedy zvlášť.

```
import androidx.room.Entity;
import androidx.room.PrimaryKey;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

@Entity(tableName = "users")
public class User implements Authenticable {

    @PrimaryKey(autoGenerate = true)
    private int id;

    @NonNull
    private String name;

    @NonNull
    private String surname;

    @NonNull
    private String email;

    @NonNull
    private String role; // "student", "employee", "visitor"

    // Specificke polia podľa role
    @Nullable
    private String faculty; // pre studentov

    @Nullable
    private String department; // pre zamestnancov

    @Nullable
    private String visitReason; // pre navstevnikov

    // Konštruktor
    public User(@NonNull String name, @NonNull String surname, @NonNull String email, @NonNull String role) {
        this.name = name;
        this.surname = surname;
        this.email = email;
        this.role = role;
    }

    // Gettery a settery
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    @NonNull
    public String getName() { return name; }
    public void setName(@NonNull String name) { this.name = name; }
```

```

@NonNull
public String getSurname() { return surname; }
public void setSurname(@NonNull String surname) { this.surname =
    surname; }

@NonNull
public String getEmail() { return email; }
public void setEmail(@NonNull String email) { this.email = email; }

@NonNull
public String getRole() { return role; }
public void setRole(@NonNull String role) { this.role = role; }

@Nullable
public String getFaculty() { return faculty; }
public void setFaculty(@Nullable String faculty) { this.faculty =
    faculty; }

@Nullable
public String getDepartment() { return department; }
public void setDepartment(@Nullable String department) { this.
    department = department; }

@Nullable
public String getVisitReason() { return visitReason; }
public void setVisitReason(@Nullable String visitReason) { this.
    visitReason = visitReason; }

// Autentifikacne metody
@Override
public void login() {
    // implementacia podla role
}

@Override
public void logout() {
    // implementacia podla role
}
}

```

Listing 50: User.java

```

import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

@Dao
public interface UserDao {

    @Insert
    void insertUser(User user);

    @Update
    void updateUser(User user);

    @Delete
    void deleteUser(User user);

    @Query("SELECT * FROM users")
    List<User> getAllUsers();

    @Query("SELECT * FROM users WHERE id = :id LIMIT 1")
    User getUserById(int id);

    @Query("SELECT * FROM users WHERE email = :email LIMIT 1")
    User getUserByEmail(String email);

    @Query("DELETE FROM users")
    void deleteAllUsers();
}

```

Listing 51: UserDao.java


```

import android.util.Log;

import com.example.parkingvsb.Authenticable;

public class SSOAuthentication implements Authenticable {

    private String token;
    private boolean isAuthenticated = false;

    @Override
    public void login() {
        Log.d("SSOAuthentication", "Spustam SSO prihlasenie...");
        if (verifyUser()) {
            generateToken();
            isAuthenticated = true;
            Log.d("SSOAuthentication", "Prihlasenie uspesne. Token: " +
                token);
        } else {
            Log.d("SSOAuthentication", "Prihlasenie zlyhalo.");
        }
    }

    @Override
    public void logout() {
        token = null;
        isAuthenticated = false;
        Log.d("SSOAuthentication", "Pouzivatel bol odhlaseny.");
    }

    public boolean verifyUser() {
        // Tu by prebehlo overenie cez externy SSO provider (napr.
        // univerzitny server)
        Log.d("SSOAuthentication", "Overujem pouzivателя cez SSO...");
        return true; // Simulacia uspechu
    }

    public void generateToken() {
        token = "SSO_TOKEN_" + System.currentTimeMillis();
    }

    public String getToken() {
        return token;
    }

    public boolean isAuthenticated() {
        return isAuthenticated;
    }
}

```

Listing 52: SSOAuthentication.java

```

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

import com.example.parkingvsb.auth.SSOAuthentication;
import com.example.parkingvsb.auth.CodeAuthentication;

public class SettingsActivity extends AppCompatActivity {

    private Button btnLogout;

    // Příklad: Uchovávame referenciu na autentifikatora (v realnej
    // app by sa to robilo cez Dependency Injection alebo ViewModel)
    private Authenticable authenticator;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        btnLogout = findViewById(R.id.btnLogout);

        // Predpokladajme, ze pouzivatel bol prihlaseny cez SSO
        // Alebo by si si mohol vytvorit staticky spravca
        // prihlasovania (napr. AuthManager), ktory drzi spravny
        // autentifikator
        authenticator = new SSOAuthentication(); // Alebo napr. new
        CodeAuthentication();

        btnLogout.setOnClickListener(v -> logout());
    }

    private void logout() {
        // Odhlásenie pouzivatela
        if (authenticator != null) {
            authenticator.logout();
        }

        // Presmerovanie spat na prihlasovaci (hlavnu) obrazovku
        Intent intent = new Intent(SettingsActivity.this,
            MainActivity.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.
            FLAG_ACTIVITY_CLEAR_TASK); // Vymaze back stack
        startActivity(intent);
        finish();
    }
}

```

Listing 53: SettingsActivity.java

```

import androidx.room.Entity;
import androidx.room.PrimaryKey;

import java.time.LocalDateTime;

@Entity(tableName = "reservations")
public class Reservation {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String userId; // ID používateľa, ktorý vykonal rezerváciu
    private int parkingSpotId; // ID parkovacieho miesta
    private LocalDateTime startTime; // Zatiaok rezervácie
    private LocalDateTime endTime; // Koniec rezervácie

    // Konštruktor
    public Reservation(String userId, int parkingSpotId,
        LocalDateTime startTime, LocalDateTime endTime) {
        this.userId = userId;
        this.parkingSpotId = parkingSpotId;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    // Gettery a settery (nutné pre Room)
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getUserId() { return userId; }
    public void setUserId(String userId) { this.userId = userId; }

    public int getParkingSpotId() { return parkingSpotId; }
    public void setParkingSpotId(int parkingSpotId) { this.parkingSpotId = parkingSpotId; }

    public LocalDateTime getStartTime() { return startTime; }
    public void setStartTime(LocalDateTime startTime) { this.startTime = startTime; }

    public LocalDateTime getEndTime() { return endTime; }
    public void setEndTime(LocalDateTime endTime) { this.endTime = endTime; }

    // Metoda na kontrolu, či môže byť rezervácia zrušená
    public boolean canBeCancelled() {
        return LocalDateTime.now().isBefore(startTime.minusMinutes(30));
    }
}

```

```

// Vytvorenie rezervacie (mozna rozsiritelna metoda)
public void create() {
    // Implementacia logiky pre vytvorenie rezervacie
}

// Zrusenie rezervacie
public void cancel() {
    // Implementacia logiky pre zrusenie rezervacie
}

// Zmena rezervacie (napr. zmena casu alebo parkovacieho miesta)
public void change(LocalDateTime newStartTime, LocalDateTime
newEndTime) {
    this.startTime = newStartTime;
    this.endTime = newEndTime;
    // Mozno aj dalsie upravu parkovacieho miesta alebo inych
    parametrov
}

@Override
public String toString() {
    return "Rezervacia: " + parkingSpotId + " (" + startTime.
        toString() + " - " + endTime.toString() + ")";
}
}

```

Listing 54: Reservation.java

```

public interface Authenticable {
    void login();
    void logout();
}

```

Listing 55: Authenticable.java

```

import android.os.Bundle;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.room.Room;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class ReservationActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private ParkingSpotAdapter adapter;
    private List<ParkingSpot> parkingSpots;
    private AppDatabase db;
    private ReservationDao reservationDao;
    private ParkingSpotDao parkingSpotDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation);

        // Inicializacia Room databazy
        db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        reservationDao = db.reservationDao();
        parkingSpotDao = db.parkingSpotDao();

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        parkingSpots = new ArrayList<>();

        // Naplnenie zoznamu parkovacich miest (toto by malo byt z
        // databazy alebo servera)
        // Tu pridame len priklad parkovacich miest
        parkingSpots.add(new ParkingSpot(1, "A1", true, "student"));
        parkingSpots.add(new ParkingSpot(2, "B2", false, "employee"));
        ;
        // Tento zoznam moze byt nacistany z databazy alebo API

        adapter = new ParkingSpotAdapter(parkingSpots, spot ->
            reserveSpot(spot));
        recyclerView.setAdapter(adapter);
    }

    private void reserveSpot(ParkingSpot spot) {

```

```

// Ak je parkovacie miesto dostupne, vytvorime rezerváciu
if (spot.isAvailable()) {
    // Vytvorenie rezervácie pre dane miesto
    Reservation reservation = new Reservation("1", spot.getId
        (), LocalDateTime.now(), LocalDateTime.now().plusHours
        (1));

    // Uložíme rezerváciu do databázy
    reservationDao.insert(reservation);

    // Oznamujeme používateľu, že rezervácia bola úspešná
    Toast.makeText(this, "Rezervácia úspešná!", Toast.
        LENGTH_SHORT).show();

    // Aktualizujeme stav parkovacieho miesta (už je obsadené
    )
    spot.setAvailable(false);
    parkingSpotDao.update(spot);

    // Obnovíme zoznam parkovacích miest, aby sa zmeny
    prejavili
    adapter.notifyDataSetChanged();
} else {
    // Ak miesto nie je dostupné
    Toast.makeText(this, "Miesto už je obsadené", Toast.
        LENGTH_SHORT).show();
}
}
}

```

Listing 56: ReservationActivity.java

```

import android.app.AlertDialog;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.util.List;

public class ReservationListActivity extends AppCompatActivity {

    private ListView listView;
    private List<Reservation> reservations;
    private AppDatabase db;
    private ReservationDao reservationDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation_list);

        listView = findViewById(R.id.listView);

        // Inicializacia Room databazy
        db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        reservationDao = db.reservationDao();

        // Nacitame vsetky rezervacie z databazy
        loadReservations();

        ArrayAdapter<Reservation> adapter = new ArrayAdapter<>(this,
            android.R.layout.simple_list_item_1, reservations);
        listView.setAdapter(adapter);

        listView.setOnItemClickListener((parent, view, position, id)
            -> {
                Reservation reservation = reservations.get(position);
                showCancelDialog(reservation);
            });
    }

    private void loadReservations() {
        // Nacitame rezervacie z databazy na hlavnom vlakne
        new Thread(() -> {
            reservations = reservationDao.getAllReservations();
            runOnUiThread(() -> {
                // Po nactani rezervacii, zaktualizujeme UI
                ArrayAdapter<Reservation> adapter = new ArrayAdapter

```

```

        <>(ReservationListActivity.this, android.R.layout.
            simple_list_item_1, reservations);
        listView.setAdapter(adapter);
    });
}).start();
}

private void showCancelDialog(Reservation reservation) {
    new AlertDialog.Builder(this)
        .setMessage("Chcete zrusit tuto rezervaciju?")
        .setPositiveButton("Zrusit", (dialog, which) ->
            cancelReservation(reservation))
        .setNegativeButton("Zrusit", null)
        .show();
}

private void cancelReservation(Reservation reservation) {
    if (reservation.canBeCancelled()) {
        // Ak rezervaciju mozno zrusit, vymazeme ju z databazy
        new Thread(() -> {
            reservationDao.delete(reservation);
            runOnUiThread(() -> {
                Toast.makeText(this, "Rezervacia zrusena!", Toast
                    .LENGTH_SHORT).show();
                loadReservations(); // Obnovime zoznam rezervacii
            });
        }).start();
    } else {
        Toast.makeText(this, "Zrusenie nie je mozne, rezervacia
            uz zacala.", Toast.LENGTH_SHORT).show();
    }
}
}
}

```

Listing 57: ReservationListActivity.java


```

import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

@Dao
public interface ReservationDao {

    @Insert
    void insert(Reservation reservation);

    @Update
    void update(Reservation reservation);

    @Delete
    void delete(Reservation reservation);

    @Query("SELECT * FROM reservations")
    List<Reservation> getAllReservations();

    @Query("SELECT * FROM reservations WHERE id = :reservationId  
LIMIT 1")
    Reservation getReservationById(int reservationId);
}

```

Listing 58: ReservationDAO.java

```

import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "parking_spots")
public class ParkingSpot {

    @PrimaryKey(autoGenerate = true)
    private int id;                // ID parkovacieho miesta (
        automaticky generovane)
    private String label;          // Oznacenie miesta, napr. "A1"
    private boolean isAvailable;   // true = volne, false = obsadene
    private String category;       // "student" alebo "employee"

    // Konstruktory
    public ParkingSpot(int id, String label, boolean isAvailable,
        String category) {
        this.id = id;
        this.label = label;
        this.isAvailable = isAvailable;
        this.category = category;
    }

    // Gettery a settery
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getLabel() {
        return label;
    }

    public void setLabel(String label) {
        this.label = label;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }

    public String getCategory() {
        return category;
    }
}

```

```

    public void setCategory(String category) {
        this.category = category;
    }

    @Override
    public String toString() {
        return label + " (" + category + ") - " + (isAvailable ? "
            Volne" : "Obsadene");
    }
}

```

Listing 59: ParkingSpot.java

```

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;
import androidx.room.Delete;

import java.util.List;

@Dao
public interface ParkingSpotDao {
    @Insert
    void insert(ParkingSpot spot);

    @Update
    void update(ParkingSpot spot);

    @Delete
    void delete(ParkingSpot spot);

    @Query("SELECT * FROM parking_spots")
    List<ParkingSpot> getAll();

    @Query("SELECT * FROM parking_spots WHERE id = :spotId LIMIT 1")
    ParkingSpot getById(int spotId);
}

```

Listing 60: ParkingSpotDAO.java

```

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.util.List;

public class ParkingSpotAdapter extends RecyclerView.Adapter<
    ParkingSpotAdapter.ViewHolder> {

    private List<ParkingSpot> spots;
    private OnSpotClickListener listener;

    public interface OnSpotClickListener {
        void onSpotClick(ParkingSpot spot);
    }

    public ParkingSpotAdapter(List<ParkingSpot> spots,
        OnSpotClickListener listener) {
        this.spots = spots;
        this.listener = listener;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
        int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(android.R.layout.simple_list_item_2, parent,
                false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int
        position) {
        ParkingSpot spot = spots.get(position);
        holder.label.setText("Miesto: " + spot.getLabel());
        holder.status.setText(spot.isAvailable() ? "Volne" : "
            Obsadene");
        holder.itemView.setOnClickListener(v -> listener.onSpotClick(
            spot));
    }

    @Override
    public int getItemCount() {
        return spots.size();
    }
}

```

```

public static class ViewHolder extends RecyclerView.ViewHolder {
    TextView label, status;

    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        label = itemView.findViewById(android.R.id.text1);
        status = itemView.findViewById(android.R.id.text2);
    }
}
}

```

Listing 61: ParkingSpotAdapter.java

```

import java.util.List;

public class ParkingLot {
    private int id;
    private String name;
    private String location;
    private int numberOfSpots;
    private List<ParkingSpot> parkingSpots;

    public ParkingLot(int id, String name, String location, List<
        ParkingSpot> parkingSpots) {
        this.id = id;
        this.name = name;
        this.location = location;
        this.parkingSpots = parkingSpots;
        this.numberOfSpots = parkingSpots.size();
    }

    public void showMap() {
        // V realnej aplikacii by sa tu mohla volat napr. MapActivity
        // alebo fragment
    }

    public List<ParkingSpot> getParkingSpots() {
        return parkingSpots;
    }

    public void setParkingSpots(List<ParkingSpot> parkingSpots) {
        this.parkingSpots = parkingSpots;
        this.numberOfSpots = parkingSpots.size();
    }
}

```

Listing 62: ParkingLot.java

```
import java.util.ArrayList;
import java.util.List;

public class ParkingSystem {
    private List<Reservation> reservations;

    public ParkingSystem() {
        reservations = new ArrayList<>();
    }

    public boolean checkAvailability(ParkingSpot spot) {
        return spot.isAvailable();
    }

    public void confirmReservation(Reservation reservation) {
        reservations.add(reservation);
    }

    public void cancelReservation(Reservation reservation) {
        reservations.remove(reservation);
    }

    public List<Reservation> getReservations() {
        return reservations;
    }
}
```

Listing 63: ParkingSystem.java

```

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.time.LocalDateTime;

public class LoginActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;
    private EditText inputVisitorCode;
    private TextView loginError; // Declare loginError TextView
    private UserDao userDao;
    private CodeDao codeDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // Inicializacia UserDao a CodeDao
        AppDatabase db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        userDao = db.userDao();
        codeDao = db.codeDao();

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);
        inputVisitorCode = findViewById(R.id.inputVisitorCode);
        loginError = findViewById(R.id.loginError); // Initialize the
            loginError TextView

        btnSSOLogin.setOnClickListener(v -> {
            // Simulovane SSO prihlasenie
            String email = "jan.novak@vsb.cz";
            User existingUser = userDao.getUserByEmail(email);

            if (existingUser == null) {
                // Ak pouzivatel neexistuje, vytvorime noveho (napr.
                studenta)
                User newUser = new User("Jan", "Novak", email, "student");
                newUser.setFaculty("Faculty of Engineering");
                userDao.insertUser(newUser);
                existingUser = userDao.getUserByEmail(email);
            }

            goToMainScreen(existingUser);
        });

        btnVisitorLogin.setOnClickListener(v -> {
            String code = inputVisitorCode.getText().toString();

```

```

// Overenie kodu v databaze
CodeAuthentication codeAuth = codeDao.getCodeByCode(code);

if (codeAuth != null && codeAuth.getExpiration().isAfter(
    LocalDateTime.now())) {
    // Ak je kod platny
    String email = "visitor@vsb.cz";
    User existingUser = userDao.getUserByEmail(email);

    if (existingUser == null) {
        // Ak pouzivatel neexistuje, vytvorime noveho
        // navstevnika
        User newUser = new User("Visitor", "Name", email, "
            visitor");
        newUser.setVisitReason("Meeting with Professor");
        userDao.insertUser(newUser);
        existingUser = userDao.getUserByEmail(email);
    }

    // Odstranime jednorazovy kod po uspesnom prihlaseni
    codeDao.deleteAllCodes();

    goToMainScreen(existingUser);
} else {
    // Zobrazenie chybovej spravy pri neplatnom kode
    loginError.setVisibility(View.VISIBLE); // Zobrazenie
    chyby
    Toast.makeText(this, "Neplatny alebo expirovany kod", Toast
        .LENGTH_SHORT).show();
}
});
}

private void goToMainScreen(User user) {
    Intent intent = new Intent(LoginActivity.this, MainActivity.class);
    intent.putExtra("userId", user.getId());
    intent.putExtra("userName", user.getName());
    intent.putExtra("userSurname", user.getSurname());
    intent.putExtra("userRole", user.getRole());
    startActivity(intent);
    finish();
}
}

```

Listing 64: LoginActivity.java


```

import androidx.room.Entity;
import androidx.room.PrimaryKey;
import java.time.LocalDateTime;

@Entity(tableName = "codes")
public class CodeAuthentication {

    @PrimaryKey(autoGenerate = true)
    private int id;
    private String code;
    private LocalDateTime expiration;

    // Konstruktor
    public CodeAuthentication(String code, LocalDateTime expiration)
    {
        this.code = code;
        this.expiration = expiration;
    }

    // Gettery a settery
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public LocalDateTime getExpiration() {
        return expiration;
    }

    public void setExpiration(LocalDateTime expiration) {
        this.expiration = expiration;
    }
}

```

Listing 65: CodeAuthentication.java

```

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

@Dao
public interface CodeDao {

    @Insert
    void insert(CodeAuthentication code);

    @Query("SELECT * FROM codes WHERE code = :code LIMIT 1")
    CodeAuthentication getCodeByCode(String code);

    @Query("DELETE FROM codes")
    void deleteAllCodes();
}

```

Listing 66: CodeDAO.java

```

import androidx.room.TypeConverter;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Converters {

    private static final DateTimeFormatter formatter =
        DateTimeFormatter.ISO_LOCAL_DATE_TIME;

    @TypeConverter
    public static String fromLocalDateTime(LocalDateTime dateTime) {
        return dateTime == null ? null : dateTime.format(formatter);
    }

    @TypeConverter
    public static LocalDateTime toLocalDateTime(String value) {
        return value == null ? null : LocalDateTime.parse(value,
            formatter);
    }
}

```

Listing 67: Converters.java

```

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.time.LocalDateTime;

public class MainActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;
    private EditText inputVisitorCode;
    private UserDao userDao;
    private CodeDao codeDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login); // Assuming
            activity_login.xml is the login screen

        // Inicializacia UserDao a CodeDao pre pristup k Room database
        AppDatabase db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        userDao = db.userDao();
        codeDao = db.codeDao();

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);
        inputVisitorCode = findViewById(R.id.inputVisitorCode);

        btnSSOLogin.setOnClickListener(v -> loginSSO());
        btnVisitorLogin.setOnClickListener(v -> loginVisitor());
    }

    private void loginSSO() {
        // Simulovane SSO prihlasenie
        String email = "jan.novak@vsb.cz";
        User existingUser = userDao.getUserByEmail(email);

        if (existingUser == null) {
            // Ak pouzivatel neexistuje, vytvorime noveho (napr. studenta)
            User newUser = new User("Jan", "Novak", email, "student");
            newUser.setFaculty("Faculty of Engineering");
            userDao.insertUser(newUser);
            existingUser = userDao.getUserByEmail(email);
        }

        goToMainScreen(existingUser);
    }

    private void loginVisitor() {
        // Spracovanie prihlasenia navstevnika pomocou kodu
        String code = inputVisitorCode.getText().toString();
    }

```

```

CodeAuthentication codeAuth = codeDao.getCodeByCode(code);

if (codeAuth != null && codeAuth.getExpiration().isAfter(
    LocalDateTime.now())) {
    // Ak je kod platny
    String email = "visitor@vsb.cz";
    User existingUser = userDao.getUserByEmail(email);

    if (existingUser == null) {
        // Ak pouzivatel neexistuje, vytvorime noveho navstevnika
        User newUser = new User("Visitor", "Name", email, "visitor"
            );
        newUser.setVisitReason("Meeting with Professor");
        userDao.insertUser(newUser);
        existingUser = userDao.getUserByEmail(email);
    }

    // Odstranime jednorazovy kod po uspesnom prihlaseni
    codeDao.deleteAllCodes();

    goToMainScreen(existingUser);
} else {
    Toast.makeText(this, "Neplatny alebo expirovany kod", Toast.
        LENGTH_SHORT).show();
}

}

private void goToMainScreen(User user) {
    // Presmerovanie na hlavny screen (napr. pre rezervaciu parkovania)
    Intent intent = new Intent(MainActivity.this, ReservationActivity.
        class);
    intent.putExtra("userId", user.getId());
    intent.putExtra("userName", user.getName());
    intent.putExtra("userSurname", user.getSurname());
    intent.putExtra("userRole", user.getRole());
    startActivity(intent);
    finish();
}
}

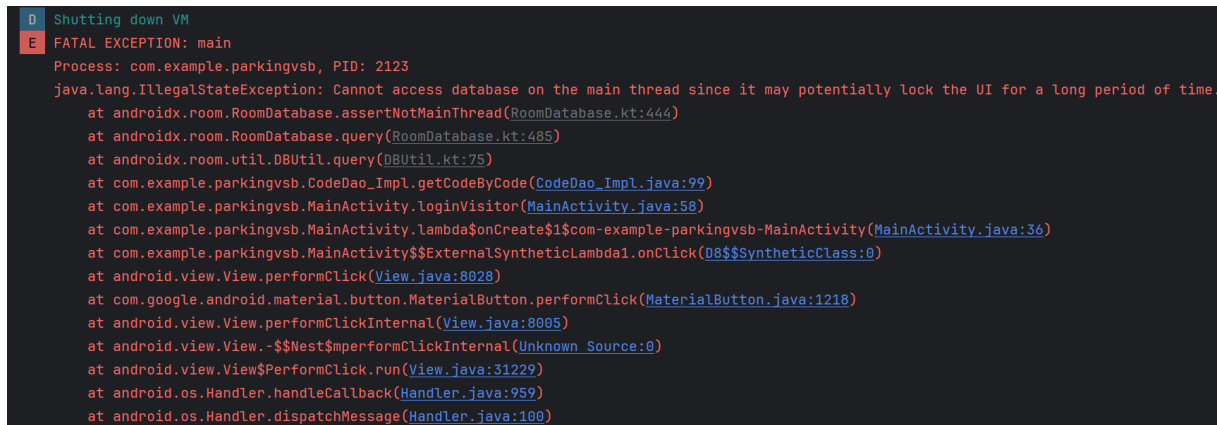
```

Listing 68: MainActivity.java

3.4 Problém za problémom

Po úspešnom buildnutí aplikácie sa objavil problém na inom mieste - po kliknutí na akékoľvek tlačidlo aplikácia spadne.

Problém je v tom, že sa pokúšame pristupovať k Room databáze z hlavného (UI) vlákna, čo Android zakazuje, aby sa zabránilo zaseknutiu aplikácie. **Čo je samozrejme vec, na ktorú ChatGPT URČITE zabudol.**



```
D Shutting down VM
E FATAL EXCEPTION: main
Process: com.example.parkingvsb, PID: 2123
java.lang.IllegalStateException: Cannot access database on the main thread since it may potentially lock the UI for a long period of time
    at androidx.room.RoomDatabase.assertNotMainThread(RoomDatabase.kt:444)
    at androidx.room.RoomDatabase.query(RoomDatabase.kt:485)
    at androidx.room.util.DBUtil.query(DBUtil.kt:75)
    at com.example.parkingvsb.CodeDao_Impl.getCodeByCode(CodeDao_Impl.java:99)
    at com.example.parkingvsb.MainActivity.loginVisitor(MainActivity.java:58)
    at com.example.parkingvsb.MainActivity.lambda$onCreate$1$com-example-parkingvsb-MainActivity(MainActivity.java:36)
    at com.example.parkingvsb.MainActivity$$ExternalSyntheticLambda1.onClick(D8$$SyntheticClass:0)
    at android.view.View.performClick(View.java:8028)
    at com.google.android.material.button.MaterialButton.performClick(MaterialButton.java:1218)
    at android.view.View.performClickInternal(View.java:8005)
    at android.view.View.$$Nest$performClickInternal(Unknown Source:0)
    at android.view.View$PerformClick.run(View.java:31229)
    at android.os.Handler.handleCallback(Handler.java:959)
    at android.os.Handler.dispatchMessage(Handler.java:100)
```

Obr. 35: Ukážka z logu Android Studia o páde aplikácie

ChatGPT nám poradil do kódov pridať Executors, ktoré slúžia na správu a spúšťanie vlákien (threads) – umožňujú jednoduchým spôsobom vykonávať asynchrónne úlohy mimo hlavného (UI) vlákna. Nižšie uvádzame len celkový opravený kód (prompt spočíval v skopírovaní erroru do ChatGPT). Taktiež sme ešte museli pridať do AndroidManifest.xml ReservationActivity.

```

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.time.LocalDateTime;
import java.util.concurrent.Executors;

public class MainActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;
    private EditText inputVisitorCode;
    private UserDao userDao;
    private CodeDao codeDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login); // Predpokladame, ze
        activity_login.xml je prihlasovacia obrazovka

        // Inicializacia UserDao a CodeDao pre pristup k Room database
        AppDatabase db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        userDao = db.userDao();
        codeDao = db.codeDao();

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);
        inputVisitorCode = findViewById(R.id.inputVisitorCode);

        btnSSOLogin.setOnClickListener(v -> loginSSO());
        btnVisitorLogin.setOnClickListener(v -> loginVisitor());
    }

    private void loginSSO() {
        Executors.newSingleThreadExecutor().execute(() -> {
            String email = "jan.novak@vsb.cz";
            User existingUser = userDao.getUserByEmail(email);

            if (existingUser == null) {
                User newUser = new User("Jan", "Novak", email, "student");
                newUser.setFaculty("Faculty of Engineering");
                userDao.insertUser(newUser);
                existingUser = userDao.getUserByEmail(email);
            }

            User finalExistingUser = existingUser;
            runOnUiThread(() -> goToMainScreen(finalExistingUser));
        });
    }

    private void loginVisitor() {

```

```

String code = inputVisitorCode.getText().toString();

Executors.newSingleThreadExecutor().execute(() -> {
    CodeAuthentication codeAuth = codeDao.getCodeByCode(code);

    if (codeAuth != null && codeAuth.getExpiration().isAfter(
        LocalDateTime.now())) {
        String email = "visitor@vsb.cz";
        User existingUser = userDao.getUserByEmail(email);

        if (existingUser == null) {
            User newUser = new User("Visitor", "Name", email, "
                visitor");
            newUser.setVisitReason("Meeting with Professor");
            userDao.insertUser(newUser);
            existingUser = userDao.getUserByEmail(email);
        }

        codeDao.deleteAllCodes();

        // Presun na dalsiu aktivitu musi byt na hlavnom vlakne
        User finalExistingUser = existingUser;
        runOnUiThread(() -> goToMainScreen(finalExistingUser));
    } else {
        runOnUiThread(() ->
            Toast.makeText(this, "Neplatny alebo expirovany kod
                ", Toast.LENGTH_SHORT).show()
        );
    }
});
}

private void goToMainScreen(User user) {
    // Presmerovanie na hlavny screen (napr. pre rezerváciu parkovania)
    Intent intent = new Intent(MainActivity.this, ReservationActivity.
        class);
    intent.putExtra("userId", user.getId());
    intent.putExtra("userName", user.getName());
    intent.putExtra("userSurname", user.getSurname());
    intent.putExtra("userRole", user.getRole());
    startActivity(intent);
    finish();
}
}

```

Listing 69: MainActivity.java

```

import android.os.Bundle;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.room.Room;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Executors;

public class ReservationActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private ParkingSpotAdapter adapter;
    private List<ParkingSpot> parkingSpots;
    private AppDatabase db;
    private ReservationDao reservationDao;
    private ParkingSpotDao parkingSpotDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation);

        // Inicializacia Room databazy
        db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        reservationDao = db.reservationDao();
        parkingSpotDao = db.parkingSpotDao();

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        parkingSpots = new ArrayList<>();

        // Naplnenie zoznamu parkovacich miest (toto by malo byt z
        // databazy alebo servera)
        // Tu pridame len priklad parkovacich miest
        parkingSpots.add(new ParkingSpot(1, "A1", true, "student"));
        parkingSpots.add(new ParkingSpot(2, "B2", false, "employee"));
        ;
        // Tento zoznam moze byt nacistany z databazy alebo API

        adapter = new ParkingSpotAdapter(parkingSpots, spot ->
            reserveSpot(spot));
        recyclerView.setAdapter(adapter);
    }
}

```



```

private void reserveSpot(ParkingSpot spot) {
    if (spot.isAvailable()) {
        Executors.newSingleThreadExecutor().execute(() -> {
            // Vytvorenie rezervacie pre dane miesto
            Reservation reservation = new Reservation("1", spot.
                getId(), LocalDateTime.now(), LocalDateTime.now().
                plusHours(1));
            reservationDao.insert(reservation);

            // Aktualizacia dostupnosti miesta
            spot.setAvailable(false);
            parkingSpotDao.update(spot);

            // UI operacie spat na hlavnom vlakne
            runOnUiThread(() -> {
                Toast.makeText(this, "Rezervacia uspesna!", Toast
                    .LENGTH_SHORT).show();
                adapter.notifyDataSetChanged();
            });
        });
    } else {
        Toast.makeText(this, "Miesto uz je obsadene", Toast.
            LENGTH_SHORT).show();
    }
}
}

```

Listing 70: ReservationActivity.java

Po týchto úpravách **APLIKÁCIA KONEČNE NIEČO ROBÍ!** Video ukážku toho, čo aplikácia aktuálne umožňuje je možné si stiahnuť z tohto odkazu.

4 Malými krôčkami k výsledku

Aktuálne máme aplikáciu v stave, že je možné sa prihlásiť pomocou SSO alebo jednorazovým kódom (predpokladáme však, že užívateľ ho zadá zlý). Následne je možné si rezervovať parkovacie miesto.

4.1 Spájame funkcionality

Máme vytvorených zopár XML aktivít, ktoré sa aktívne nezobrazujú a nepoužívajú v aplikácii - `activity_parking_map` a `activity_settings`, `dialog_cancel_reservation`. Pokúsime sa tieto aktivity „priviesť k životu“. Zároveň sme zabudli na jednu podstatnú funkcionality - výber časového intervalu rezervácie. Takže sa pokúsime implementovať aj ju.

Nakoniec po dlhom hadaní s ChatGPT, ktoré tu pre zbytočné nezahlcovanie tohto dokumentu neuvádzame, sme v aktuálnej verzii aplikácie sfunkčnili iba výber času a zobrazenie mapy. Nižšie zároveň uvádzame všetky kompletne zdrojové kódy aplikácie, ktoré je možné zároveň nájsť aj v GitHub repozitári projektu.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/vsb"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ParkingVSB"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ReservationActivity" />
        <activity android:name=".LoginActivity" />
        <activity android:name=".MapActivity"
            android:exported="true"
            android:label="MapActivity">
        </activity>
        <activity android:name=".SettingsActivity" android:exported="true"
        />
    </application>

</manifest>
```

Listing 71: AndroidManifest.xml

```

package com.example.parkingvsb;

import android.content.Context;
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import androidx.room.TypeConverters;
import androidx.room.migration.Migration;
import androidx.sqlite.db.SupportSQLiteDatabase;

import java.util.concurrent.Executors;

@Database(
    entities = {
        User.class,
        Reservation.class,
        ParkingSpot.class,
        CodeAuthentication.class
    },
    version = 2 // Verzia databazy
)
@TypeConverters({Converters.class})
public abstract class AppDatabase extends RoomDatabase {

    public abstract UserDao userDao();
    public abstract ReservationDao reservationDao();
    public abstract ParkingSpotDao parkingSpotDao();
    public abstract CodeDao codeDao();

    private static volatile AppDatabase instance;

    public static synchronized AppDatabase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(
                context.getApplicationContext(),
                AppDatabase.class,
                "parking_database"
            )
                .addMigrations(MIGRATION_1_2) // Pridanie migracie
                .fallbackToDestructiveMigration() // Toto sa moze
                pouzit na vymazanie databazy, ak migracia nie je k
                dispozicii.
                .addCallback(new RoomDatabase.Callback() {
                    @Override
                    public void onCreate(@NonNull SupportSQLiteDatabase
                        db) {
                        super.onCreate(db);
                        // Zavolanie metody na pridanie testovacich dat
                        po vytvoreni databazy
                        Executors.newSingleThreadExecutor().execute(()
                            -> populateTestData(context));
                    }
                })
                .build();
        }
    }
}

```

```

    }
    return instance;
}

// Priklad migracneho skriptu medzi verzia 1 a 2
static final Migration MIGRATION_1_2 = new Migration(1, 2) {
    @Override
    public void migrate(@NonNull SupportSQLiteDatabase database) {
        // V migracii pridame nove stlpce pre GPS suradnice
        database.execSQL("ALTER TABLE parking_spots ADD COLUMN
            location_lat DOUBLE NOT NULL DEFAULT 0");
        database.execSQL("ALTER TABLE parking_spots ADD COLUMN
            location_long DOUBLE NOT NULL DEFAULT 0");
    }
};

// Metoda na pridanie testovacich dat do databazy
private static void populateTestData(Context context) {
    AppDatabase db = AppDatabase.getInstance(context);
    Executors.newSingleThreadExecutor().execute(() -> {
        // Skontroluj, ci je tabulka parkovacich miest prazdna
        if (db.parkingSpotDao().getAllParkingSpots().isEmpty()) {
            Log.d("AppDatabase", "Pridavam testovacie data do databazy");
            db.parkingSpotDao().insert(new ParkingSpot(0, "A1", true, "
                student", 48.148, 17.107));
            db.parkingSpotDao().insert(new ParkingSpot(0, "B2", false, "
                employee", 48.1485, 17.1075));
            db.parkingSpotDao().insert(new ParkingSpot(0, "C3", true, "
                student", 48.149, 17.108));
        } else {
            Log.d("AppDatabase", "Tabulka uz obsahuje data");
        }
    });
}
}
}

```

Listing 72: AppDatabase.java

```

public interface Authenticable {
    void login();
    void logout();
}

```

Listing 73: Authenticable.java

```

package com.example.parkingvsb;

import androidx.room.Entity;
import androidx.room.PrimaryKey;
import java.time.LocalDateTime;

@Entity(tableName = "codes")
public class CodeAuthentication {

    @PrimaryKey(autoGenerate = true)
    private int id;
    private String code;
    private LocalDateTime expiration;

    // Konstruktor
    public CodeAuthentication(String code, LocalDateTime expiration) {
        this.code = code;
        this.expiration = expiration;
    }

    // Gettery a settery
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public LocalDateTime getExpiration() {
        return expiration;
    }

    public void setExpiration(LocalDateTime expiration) {
        this.expiration = expiration;
    }
}

```

Listing 74: CodeAuthentication.java

```

package com.example.parkingvsb;

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

@Dao
public interface CodeDao {

    @Insert
    void insert(CodeAuthentication code);

    @Query("SELECT * FROM codes WHERE code = :code LIMIT 1")
    CodeAuthentication getCodeByCode(String code);

    @Query("DELETE FROM codes")
    void deleteAllCodes();
}

```

Listing 75: CodeDAO.java

```

package com.example.parkingvsb;

import androidx.room.TypeConverter;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Converters {

    private static final DateTimeFormatter formatter = DateTimeFormatter.
        ISO_LOCAL_DATE_TIME;

    @TypeConverter
    public static String fromLocalDateTime(LocalDateTime dateTime) {
        return dateTime == null ? null : dateTime.format(formatter);
    }

    @TypeConverter
    public static LocalDateTime toLocalDateTime(String value) {
        return value == null ? null : LocalDateTime.parse(value, formatter);
    }
}

```

Listing 76: Converters.java

```

package com.example.parkingvsb;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.time.LocalDateTime;

public class LoginActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;
    private EditText inputVisitorCode;
    private TextView loginError; // Declare loginError TextView
    private UserDao userDao;
    private CodeDao codeDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // Inicializacia UserDao a CodeDao
        AppDatabase db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        userDao = db.userDao();
        codeDao = db.codeDao();

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);
        inputVisitorCode = findViewById(R.id.inputVisitorCode);
        loginError = findViewById(R.id.loginError); // Initialize the
            loginError TextView

        btnSSOLogin.setOnClickListener(v -> {
            // Simulovane SSO prihlasenie
            String email = "jan.novak@vsb.cz";
            User existingUser = userDao.getUserByEmail(email);

            if (existingUser == null) {
                // Ak pouzivatel neexistuje, vytvorime noveho (napr.
                studenta)
                User newUser = new User("Jan", "Novak", email, "student");
                newUser.setFaculty("Faculty of Engineering");
                userDao.insertUser(newUser);
                existingUser = userDao.getUserByEmail(email);
            }

            goToMainScreen(existingUser);
        });
    }

```

```

btnVisitorLogin.setOnClickListener(v -> {
    String code = inputVisitorCode.getText().toString();
    // Overenie kodu v databaze
    CodeAuthentication codeAuth = codeDao.getCodeByCode(code);

    if (codeAuth != null && codeAuth.getExpiration().isAfter(
        LocalDateTime.now())) {
        // Ak je kod platny
        String email = "visitor@vsb.cz";
        User existingUser = userDao.getUserByEmail(email);

        if (existingUser == null) {
            // Ak pouzivatel neexistuje, vytvorime noveho
            // navstevnika
            User newUser = new User("Visitor", "Name", email, "
                visitor");
            newUser.setVisitReason("Meeting with Professor");
            userDao.insertUser(newUser);
            existingUser = userDao.getUserByEmail(email);
        }

        // Odstranime jednorazovy kod po uspesnom prihlaseni
        codeDao.deleteAllCodes();

        goToMainScreen(existingUser);
    } else {
        // Zobrazenie chybovej spravy pri neplatnom kode
        loginError.setVisibility(View.VISIBLE); // Zobrazenie
        chyby
        Toast.makeText(this, "Neplatny alebo expirovany kod", Toast
            .LENGTH_SHORT).show();
    }
});
}

private void goToMainScreen(User user) {
    Intent intent = new Intent(LoginActivity.this, MainActivity.class);
    intent.putExtra("userId", user.getId());
    intent.putExtra("userName", user.getName());
    intent.putExtra("userSurname", user.getSurname());
    intent.putExtra("userRole", user.getRole());
    startActivity(intent);
    finish();
}
}

```

Listing 77: LoginActivity.java


```

package com.example.parkingvsb;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.time.LocalDateTime;
import java.util.concurrent.Executors;

public class MainActivity extends AppCompatActivity {

    private Button btnSSOLogin, btnVisitorLogin;
    private EditText inputVisitorCode;
    private UserDao userDao;
    private CodeDao codeDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login); // Assuming
            activity_login.xml is the login screen

        // Inicializacia UserDao a CodeDao pre pristup k Room database
        AppDatabase db = Room.databaseBuilder(getApplicationContext(),
            AppDatabase.class, "app_database").build();
        userDao = db.userDao();
        codeDao = db.codeDao();

        btnSSOLogin = findViewById(R.id.btnSSOLogin);
        btnVisitorLogin = findViewById(R.id.btnVisitorLogin);
        inputVisitorCode = findViewById(R.id.inputVisitorCode);

        btnSSOLogin.setOnClickListener(v -> loginSSO());
        btnVisitorLogin.setOnClickListener(v -> loginVisitor());
    }

    private void loginSSO() {
        Executors.newSingleThreadExecutor().execute(() -> {
            String email = "jan.novak@vsb.cz";
            User existingUser = userDao.getUserByEmail(email);

            if (existingUser == null) {
                User newUser = new User("Jan", "Novak", email, "student");
                newUser.setFaculty("Faculty of Engineering");
                userDao.insertUser(newUser);
                existingUser = userDao.getUserByEmail(email);
            }

            User finalExistingUser = existingUser;
            runOnUiThread(() -> goToMainScreen(finalExistingUser));
        });
    }
}

```

```

private void loginVisitor() {
    String code = inputVisitorCode.getText().toString();

    Executors.newSingleThreadExecutor().execute(() -> {
        CodeAuthentication codeAuth = codeDao.getCodeByCode(code);

        if (codeAuth != null && codeAuth.getExpiration().isAfter(
            LocalDateTime.now())) {
            String email = "visitor@vsb.cz";
            User existingUser = userDao.getUserByEmail(email);

            if (existingUser == null) {
                User newUser = new User("Visitor", "Name", email, "
                    visitor");
                newUser.setVisitReason("Meeting with Professor");
                userDao.insertUser(newUser);
                existingUser = userDao.getUserByEmail(email);
            }

            codeDao.deleteAllCodes();

            // Presun na dalsiu aktivitu musi byt na hlavnom vlakne
            User finalExistingUser = existingUser;
            runOnUiThread(() -> goToMainScreen(finalExistingUser));
        } else {
            runOnUiThread(() ->
                Toast.makeText(this, "Neplatny alebo expirovany kod
                    ", Toast.LENGTH_SHORT).show()
            );
        }
    });
}

private void goToMainScreen(User user) {
    // Presmerovanie na hlavny screen (napr. pre rezerváciu parkovania)
    Intent intent = new Intent(MainActivity.this, ReservationActivity.
        class);
    intent.putExtra("userId", user.getId());
    intent.putExtra("userName", user.getName());
    intent.putExtra("userSurname", user.getSurname());
    intent.putExtra("userRole", user.getRole());
    startActivity(intent);
    finish();
}
}

```

Listing 78: MainActivity.java

```

package com.example.parkingvsb;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import java.util.List;

public class MapActivity extends AppCompatActivity implements
    OnMapReadyCallback {

    private MapView mapView;
    private GoogleMap googleMap;
    private ListView parkingSpotsListView;
    private List<ParkingSpot> parkingSpots;
    private String selectedSpotId;

    private static final String MAP_VIEW_BUNDLE_KEY = "MapViewBundleKey";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.map_activity);

        // Toolbar
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        // Views
        mapView = findViewById(R.id.mapView);
        parkingSpotsListView = findViewById(R.id.parkingSpotsListView);

        // Initialize map
        Bundle mapViewBundle = null;
        if (savedInstanceState != null) {
            mapViewBundle = savedInstanceState.getBundle(
                MAP_VIEW_BUNDLE_KEY);
        }
        mapView.onCreate(mapViewBundle);
        mapView.getMapAsync(this);

        loadParkingSpots();
    }

```

```

        parkingSpotsListView.setOnItemClickListener((parent, view, position
        , id) -> {
            ParkingSpot selectedSpot = parkingSpots.get(position);
            selectedSpotId = String.valueOf(selectedSpot.getId());
            Toast.makeText(this, "Vybrali ste: " + selectedSpot.getLabel(),
                Toast.LENGTH_SHORT).show();
        });
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        this.googleMap = googleMap;

        if (parkingSpots != null) {
            displayMarkers();
        }

        googleMap.setOnMarkerClickListener(marker -> {
            ParkingSpot selectedSpot = findSpotByMarker(marker);
            if (selectedSpot != null) {
                selectedSpotId = String.valueOf(selectedSpot.getId());
                Toast.makeText(this, "Vybrali ste: " + selectedSpot.
                    getLabel(), Toast.LENGTH_SHORT).show();
            }
            return false;
        });
    }

    private void loadParkingSpots() {
        AppDatabase db = AppDatabase.getInstance(getApplicationContext());

        new Thread(() -> {
            parkingSpots = db.parkingSpotDao().getAllParkingSpots();

            runOnUiThread(() -> {
                ArrayAdapter<ParkingSpot> adapter = new ArrayAdapter<>(
                    MapActivity.this,
                    android.R.layout.simple_list_item_1,
                    parkingSpots
                );
                parkingSpotsListView.setAdapter(adapter);

                if (googleMap != null) {
                    displayMarkers();
                }
            });
        }).start();
    }

    private void displayMarkers() {
        googleMap.clear();
        for (ParkingSpot spot : parkingSpots) {
            LatLng location = new LatLng(spot.getLatitude(), spot.
                getLongitude());
            googleMap.addMarker(new MarkerOptions().position(location).
                title(spot.getLabel()));
        }
    }

```

```

LatLng cityCenter = new LatLng(48.148, 17.107);
googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(cityCenter,
    13));

googleMap.setOnMarkerClickListener(marker -> {
    ParkingSpot selectedSpot = findSpotByMarker(marker);
    if (selectedSpot != null) {
        selectedSpotId = String.valueOf(selectedSpot.getId());
        Toast.makeText(this, "Vybrali ste: " + selectedSpot.
            getLabel(), Toast.LENGTH_SHORT).show();
    }
    return false;
});
}

private ParkingSpot findSpotByMarker(Marker marker) {
    for (ParkingSpot spot : parkingSpots) {
        if (spot.getLabel().equals(marker.getTitle())) {
            return spot;
        }
    }
    return null;
}

public void onReserveClick(android.view.View view) {
    if (selectedSpotId != null) {
        Intent intent = new Intent(this, ReservationActivity.class);
        intent.putExtra("selectedSpotId", selectedSpotId);
        startActivity(intent);
    } else {
        Toast.makeText(this, "Vyberte miesto na mape alebo zozname.",
            Toast.LENGTH_SHORT).show();
    }
}

// ===== MENU (nastavenia / odhlasenie) =====
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.map_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.action_settings) {
        startActivity(new Intent(this, SettingsActivity.class));
        return true;
    }
    return super.onOptionsItemSelected(item);
}

// ===== Map lifecycle =====
@Override
protected void onResume() {
    super.onResume();
    mapView.onResume();
}

```

```

@Override
protected void onStart() {
    super.onStart();
    mapView.onStart();
}

@Override
protected void onStop() {
    super.onStop();
    mapView.onStop();
}

@Override
protected void onPause() {
    mapView.onPause();
    super.onPause();
}

@Override
protected void onDestroy() {
    mapView.onDestroy();
    super.onDestroy();
}

@Override
public void onLowMemory() {
    super.onLowMemory();
    mapView.onLowMemory();
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    Bundle mapViewBundle = outState.getBundle(MAP_VIEW_BUNDLE_KEY);
    if (mapViewBundle == null) {
        mapViewBundle = new Bundle();
        outState.putBundle(MAP_VIEW_BUNDLE_KEY, mapViewBundle);
    }
    mapView.onSaveInstanceState(mapViewBundle);
}
}

```

Listing 79: MapActivity.java

```

package com.example.parkingvsb;

import java.util.List;

public class ParkingLot {
    private int id;
    private String name;
    private String location;
    private int numberOfSpots;
    private List<ParkingSpot> parkingSpots;

    public ParkingLot(int id, String name, String location, List<
        ParkingSpot> parkingSpots) {
        this.id = id;
        this.name = name;
        this.location = location;
        this.parkingSpots = parkingSpots;
        this.numberOfSpots = parkingSpots.size();
    }

    public void showMap() {
        // V realnej aplikacii by sa tu mohla volat napr. MapActivity alebo
        fragment
    }

    public List<ParkingSpot> getParkingSpots() {
        return parkingSpots;
    }

    public void setParkingSpots(List<ParkingSpot> parkingSpots) {
        this.parkingSpots = parkingSpots;
        this.numberOfSpots = parkingSpots.size();
    }
}

```

Listing 80: ParkingLot.java

```

package com.example.parkingvsb;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "parking_spots")
public class ParkingSpot {

    @PrimaryKey(autoGenerate = true)
    private int id;           // ID parkovacieho miesta (automaticky
                             // generovane)
    private String label;     // Oznacenie miesta, napr. "A1"
    private boolean isAvailable; // true = volne, false = obsadene
    private String category;  // "student" alebo "employee"
    private double latitude;  // Zemepisna sirka parkovacieho miesta
    private double longitude; // Zemepisna dlzka parkovacieho miesta

    // Konstruktory
    public ParkingSpot(int id, String label, boolean isAvailable, String
        category, double latitude, double longitude) {
        this.id = id;
        this.label = label;
        this.isAvailable = isAvailable;
        this.category = category;
        this.latitude = latitude;
        this.longitude = longitude;
    }

    // Gettery a settery
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getLabel() {
        return label;
    }

    public void setLabel(String label) {
        this.label = label;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }

    public String getCategory() {
        return category;
    }
}

```



```

    public void setCategory(String category) {
        this.category = category;
    }

    public double getLatitude() {
        return latitude;
    }

    public void setLatitude(double latitude) {
        this.latitude = latitude;
    }

    public double getLongitude() {
        return longitude;
    }

    public void setLongitude(double longitude) {
        this.longitude = longitude;
    }

    @Override
    public String toString() {
        return label + " (" + category + ") - " + (isAvailable ? "Volne" :
            "Obsadene");
    }
}

```

Listing 81: ParkingSpot.java

```

package com.example.parkingvsb;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.util.List;

public class ParkingSpotAdapter extends RecyclerView.Adapter<
    ParkingSpotAdapter.ViewHolder> {

    private List<ParkingSpot> spots;
    private OnSpotClickListener listener;

    public interface OnSpotClickListener {
        void onSpotClick(ParkingSpot spot);
    }

    public ParkingSpotAdapter(List<ParkingSpot> spots, OnSpotClickListener
        listener) {
        this.spots = spots;
        this.listener = listener;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
        viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(android.R.layout.simple_list_item_2, parent, false
            );
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position)
    {
        ParkingSpot spot = spots.get(position);
        holder.label.setText("Miesto: " + spot.getLabel());
        holder.status.setText(spot.isAvailable() ? "Volne" : "Obsadene");
        holder.itemView.setOnClickListener(v -> listener.onSpotClick(spot))
        ;
    }

    @Override
    public int getItemCount() {
        return spots.size();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView label, status;

        public ViewHolder(@NonNull View itemView) {

```

```

        super(itemView);
        label = itemView.findViewById(android.R.id.text1);
        status = itemView.findViewById(android.R.id.text2);
    }
}

```

Listing 82: ParkingSpotAdapter.java

```

package com.example.parkingvsb;

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;
import androidx.room.Delete;

import java.util.List;

@Dao
public interface ParkingSpotDao {
    @Insert
    void insert(ParkingSpot spot);

    @Update
    void update(ParkingSpot spot);

    @Delete
    void delete(ParkingSpot spot);

    @Query("SELECT * FROM parking_spots")
    List<ParkingSpot> getAllParkingSpots();

    @Query("SELECT * FROM parking_spots WHERE id = :spotId LIMIT 1")
    ParkingSpot getById(int spotId);
}

```

Listing 83: ParkingSpotDAO.java

```
package com.example.parkingvsb;

import java.util.ArrayList;
import java.util.List;

public class ParkingSystem {
    private List<Reservation> reservations;

    public ParkingSystem() {
        reservations = new ArrayList<>();
    }

    public boolean checkAvailability(ParkingSpot spot) {
        return spot.isAvailable();
    }

    public void confirmReservation(Reservation reservation) {
        reservations.add(reservation);
    }

    public void cancelReservation(Reservation reservation) {
        reservations.remove(reservation);
    }

    public List<Reservation> getReservations() {
        return reservations;
    }
}
```

Listing 84: ParkingSystem.java

```

package com.example.parkingvsb;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

import java.time.LocalDateTime;

@Entity(tableName = "reservations")
public class Reservation {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String userId; // ID používateľa, ktorý vykonal rezerváciu
    private int parkingSpotId; // ID parkovacieho miesta
    private LocalDateTime startTime; // Začiatok rezervácie
    private LocalDateTime endTime; // Koniec rezervácie

    // Konštruktor
    public Reservation(String userId, int parkingSpotId, LocalDateTime
        startTime, LocalDateTime endTime) {
        this.userId = userId;
        this.parkingSpotId = parkingSpotId;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    // Gettery a settery (nutné pre Room)
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getUserId() { return userId; }
    public void setUserId(String userId) { this.userId = userId; }

    public int getParkingSpotId() { return parkingSpotId; }
    public void setParkingSpotId(int parkingSpotId) { this.parkingSpotId =
        parkingSpotId; }

    public LocalDateTime getStartTime() { return startTime; }
    public void setStartTime(LocalDateTime startTime) { this.startTime =
        startTime; }

    public LocalDateTime getEndTime() { return endTime; }
    public void setEndTime(LocalDateTime endTime) { this.endTime = endTime;
    }

    // Metoda na kontrolu, či môže byť rezervácia zrušená
    public boolean canBeCancelled() {
        return LocalDateTime.now().isBefore(startTime.minusMinutes(30));
    }

    // Vytvorenie rezervácie (možno rozšíriteľná metóda)
    public void create() {
        // Implementácia logiky pre vytvorenie rezervácie
    }

    // Zrušenie rezervácie

```

```

public boolean cancel(ReservationDao reservationDao, ParkingSpotDao
parkingSpotDao) {
    if (!canBeCancelled()) return false;

    reservationDao.delete(this);
    ParkingSpot spot = parkingSpotDao.getById(this.parkingSpotId);
    if (spot != null) {
        spot.setAvailable(true);
        parkingSpotDao.update(spot);
    }
    return true;
}

// Zmena rezervacie (napr. zmena casu alebo parkovacieho miesta)
public void change(LocalDateTime newStartTime, LocalDateTime newEndTime
) {
    this.startTime = newStartTime;
    this.endTime = newEndTime;
    // Mozno aj dalsie upravy parkovacieho miesta alebo inych
    parametrov
}

@Override
public String toString() {
    return "Rezervacia: " + parkingSpotId + " (" + startTime.toString()
        + " - " + endTime.toString() + ")";
}
}

```

Listing 85: Reservation.java

```

package com.example.parkingvsb;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

import java.time.LocalDateTime;

@Entity(tableName = "reservations")
public class Reservation {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String userId; // ID používateľa, ktorý vykonal rezerváciu
    private int parkingSpotId; // ID parkovacieho miesta
    private LocalDateTime startTime; // Zatiaťok rezervácie
    private LocalDateTime endTime; // Koniec rezervácie

    // Konštruktor
    public Reservation(String userId, int parkingSpotId, LocalDateTime
        startTime, LocalDateTime endTime) {
        this.userId = userId;
        this.parkingSpotId = parkingSpotId;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    // Gettery a settery (nutné pre Room)
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getUserId() { return userId; }
    public void setUserId(String userId) { this.userId = userId; }

    public int getParkingSpotId() { return parkingSpotId; }
    public void setParkingSpotId(int parkingSpotId) { this.parkingSpotId =
        parkingSpotId; }

    public LocalDateTime getStartTime() { return startTime; }
    public void setStartTime(LocalDateTime startTime) { this.startTime =
        startTime; }

    public LocalDateTime getEndTime() { return endTime; }
    public void setEndTime(LocalDateTime endTime) { this.endTime = endTime;
    }

    // Metóda na kontrolu, či môže byť rezervácia zrušená
    public boolean canBeCancelled() {
        return LocalDateTime.now().isBefore(startTime.minusMinutes(30));
    }

    // Vytvorenie rezervácie (možno rozšíriteľná metóda)
    public void create() {
        // Implementácia logiky pre vytvorenie rezervácie
    }

    // Zrušenie rezervácie

```

```

public boolean cancel(ReservationDao reservationDao, ParkingSpotDao
parkingSpotDao) {
    if (!canBeCancelled()) return false;

    reservationDao.delete(this);
    ParkingSpot spot = parkingSpotDao.getById(this.parkingSpotId);
    if (spot != null) {
        spot.setAvailable(true);
        parkingSpotDao.update(spot);
    }
    return true;
}

// Zmena rezervacie (napr. zmena casu alebo parkovacieho miesta)
public void change(LocalDateTime newStartTime, LocalDateTime newEndTime
) {
    this.startTime = newStartTime;
    this.endTime = newEndTime;
    // Mozno aj dalsie upravy parkovacieho miesta alebo inych
    parametrov
}

@Override
public String toString() {
    return "Rezervacia: " + parkingSpotId + " (" + startTime.toString()
        + " - " + endTime.toString() + ")";
}
}

```

Listing 86: ReservationActivity.java


```

package com.example.parkingvsb;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.TextView;

import java.util.List;

public class ReservationAdapter extends ArrayAdapter<Reservation> {

    private final String currentUserId;
    private final CancellableListener listener;

    public interface CancellableListener {
        void onCancelClicked(Reservation reservation);
    }

    public ReservationAdapter(Context context, List<Reservation>
        reservations, String currentUserId, CancellableListener listener) {
        super(context, 0, reservations);
        this.currentUserId = currentUserId;
        this.listener = listener;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Reservation reservation = getItem(position);

        if (convertView == null) {
            convertView = LayoutInflater.from(getContext()).inflate(R.
                layout.list_item_reservation, parent, false);
        }

        TextView reservationText = convertView.findViewById(R.id.
            reservationText);
        Button cancelButton = convertView.findViewById(R.id.cancelButton);

        reservationText.setText(reservation.toString());

        // Zobraz tlačidlo iba ak rezervacia patri pouzivatelovi
        if (reservation.getUserId().equals(currentUserId) && reservation.
            canBeCancelled()) {
            cancelButton.setVisibility(View.VISIBLE);
            cancelButton.setOnClickListener(v -> listener.onCancelClicked(
                reservation));
        } else {
            cancelButton.setVisibility(View.GONE);
        }

        return convertView;
    }
}

```

Listing 87: ReservationAdapter.java

```
package com.example.parkingvsb;

import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

@Dao
public interface ReservationDao {

    @Insert
    void insert(Reservation reservation);

    @Update
    void update(Reservation reservation);

    @Delete
    void delete(Reservation reservation);

    @Query("SELECT * FROM reservations")
    List<Reservation> getAllReservations();

    @Query("SELECT * FROM reservations WHERE id = :reservationId LIMIT 1")
    Reservation getReservationById(int reservationId);

    @Query("SELECT * FROM reservations WHERE userId = :userId")
    List<Reservation> getReservationsForUser(String userId);
}
```

Listing 88: ReservationDAO.java

```

package com.example.parkingvsb;

import android.app.AlertDialog;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.room.Room;

import java.util.List;

public class ReservationListActivity extends AppCompatActivity {

    private ListView listView;
    private List<Reservation> reservations;
    private AppDatabase db;
    private ReservationDao reservationDao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reservation_list);

        listView = findViewById(R.id.listView);

        // Inicializacia Room databazy
        db = Room.databaseBuilder(getApplicationContext(), AppDatabase.
            class, "app_database").build();
        reservationDao = db.reservationDao();

        // Nacitajme vsetky rezervacie z databazy
        loadReservations();

        ArrayAdapter<Reservation> adapter = new ArrayAdapter<>(this,
            android.R.layout.simple_list_item_1, reservations);
        listView.setAdapter(adapter);

        listView.setOnItemClickListener((parent, view, position, id) -> {
            Reservation reservation = reservations.get(position);
            showCancelDialog(reservation);
        });
    }

    private void loadReservations() {
        // Nacitajme rezervacie z databazy na hlavnom vlakne
        new Thread(() -> {
            reservations = reservationDao.getAllReservations();
            runOnUiThread(() -> {
                // Po nacisti rezervacii, zaktualizujeme UI
                ArrayAdapter<Reservation> adapter = new ArrayAdapter<>(
                    ReservationListActivity.this, android.R.layout.
                    simple_list_item_1, reservations);
                listView.setAdapter(adapter);
            });
        }).start();
    }
}

```

```

}

private void showCancelDialog(Reservation reservation) {
    new AlertDialog.Builder(this)
        .setMessage("Chcete zrusit tuto rezervaciu?")
        .setPositiveButton("Zrusit", (dialog, which) ->
            cancelReservation(reservation))
        .setNegativeButton("Zrusit", null)
        .show();
}

private void cancelReservation(Reservation reservation) {
    if (reservation.canBeCancelled()) {
        // Ak rezervaciu mozno zrusit, vymazeme ju z databazy
        new Thread(() -> {
            reservationDao.delete(reservation);
            runOnUiThread(() -> {
                Toast.makeText(this, "Rezervacia zrusena!", Toast.
                    LENGTH_SHORT).show();
                loadReservations(); // Obnovime zoznam rezervacii
            });
        }).start();
    } else {
        Toast.makeText(this, "Zrusenie nie je mozne, rezervacia uz
            zacala.", Toast.LENGTH_SHORT).show();
    }
}
}

```

Listing 89: ReservationListActivity.java

```

package com.example.parkingvsb;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

public class SettingsActivity extends AppCompatActivity {

    private Button btnLogout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings); // Uisti sa, ze nazov
            layoutu sedi

        btnLogout = findViewById(R.id.btnLogout);

        btnLogout.setOnClickListener(v -> {
            // Tu mozes vymazat SharedPreferences ak ich pouzuvas na
            // ulozenie stavu prihlasenia
            getSharedPreferences("user_prefs", MODE_PRIVATE)
                .edit()
                .clear()
                .apply();

            // Spusti MainActivity (prihlasovanie)
            Intent intent = new Intent(SettingsActivity.this, MainActivity.
                class);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.
                FLAG_ACTIVITY_CLEAR_TASK); // Vymaze backstack
            startActivity(intent);
        });
    }
}

```

Listing 90: SettingsActivity.java

```

package com.example.parkingvsb;

import android.util.Log;

public class SSOAuthentication implements Authenticable {

    private String token;
    private boolean isAuthenticated = false;

    @Override
    public void login() {
        Log.d("SSOAuthentication", "Spustam SSO prihlasenie...");
        if (verifyUser()) {
            generateToken();
            isAuthenticated = true;
            Log.d("SSOAuthentication", "Prihlasenie uspesne. Token: " +
                token);
        } else {
            Log.d("SSOAuthentication", "Prihlasenie zlyhalo.");
        }
    }

    @Override
    public void logout() {
        token = null;
        isAuthenticated = false;
        Log.d("SSOAuthentication", "Pouzivatel bol odhlaseny.");
    }

    public boolean verifyUser() {
        // Tu by prebehlo overenie cez externy SSO provider (napr.
        // univerzitny server)
        Log.d("SSOAuthentication", "Overujem pouzivателя cez SSO...");
        return true; // Simulacia uspechu
    }

    public void generateToken() {
        // Simulacia tokenu v realnej aplikacii by bol ziskany zo
        // servera
        token = "SSO_TOKEN_" + System.currentTimeMillis();
    }

    public String getToken() {
        return token;
    }

    public boolean isAuthenticated() {
        return isAuthenticated;
    }
}

```

Listing 91: SSOAuthentication.java

```

package com.example.parkingvsb;

import androidx.room.Entity;
import androidx.room.PrimaryKey;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

@Entity(tableName = "users")
public class User implements Authenticable {

    @PrimaryKey(autoGenerate = true)
    private int id;

    @NonNull
    private String name;

    @NonNull
    private String surname;

    @NonNull
    private String email;

    @NonNull
    private String role; // "student", "employee", "visitor"

    // Specifické polia podľa role
    @Nullable
    private String faculty; // pre studentov

    @Nullable
    private String department; // pre zamestnancov

    @Nullable
    private String visitReason; // pre navštevníkov

    // Konštruktor
    public User(@NonNull String name, @NonNull String surname, @NonNull
        String email, @NonNull String role) {
        this.name = name;
        this.surname = surname;
        this.email = email;
        this.role = role;
    }

    // Gettery a settery
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    @NonNull
    public String getName() { return name; }
    public void setName(@NonNull String name) { this.name = name; }

    @NonNull
    public String getSurname() { return surname; }
    public void setSurname(@NonNull String surname) { this.surname =
        surname; }

```

```

@NonNull
public String getEmail() { return email; }
public void setEmail(@NonNull String email) { this.email = email; }

@NonNull
public String getRole() { return role; }
public void setRole(@NonNull String role) { this.role = role; }

@Nullable
public String getFaculty() { return faculty; }
public void setFaculty(@Nullable String faculty) { this.faculty =
    faculty; }

@Nullable
public String getDepartment() { return department; }
public void setDepartment(@Nullable String department) { this.
    department = department; }

@Nullable
public String getVisitReason() { return visitReason; }
public void setVisitReason(@Nullable String visitReason) { this.
    visitReason = visitReason; }

// Autentifikacne metody
@Override
public void login() {
    // implementacia podla role
}

@Override
public void logout() {
    // implementacia podla role
}
}

```

Listing 92: User.java


```

package com.example.parkingvsb;

import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

@Dao
public interface UserDao {

    @Insert
    void insertUser(User user);

    @Update
    void updateUser(User user);

    @Delete
    void deleteUser(User user);

    @Query("SELECT * FROM users")
    List<User> getAllUsers();

    @Query("SELECT * FROM users WHERE id = :id LIMIT 1")
    User getUserById(int id);

    @Query("SELECT * FROM users WHERE email = :email LIMIT 1")
    User getUserByEmail(String email);

    @Query("DELETE FROM users")
    void deleteAllUsers();
}

```

Listing 93: UserDAO.java

5 Záver

Môžeme povedať, že nejako sa nám podarilo aplikáciu vytvoriť. Video funkčnosti aktuálnej verzie je možné pozrieť na tomto odkaze. Bol to síce boj, hádky o tom, že prečo ChatGPT spravil to takto a hento zas tak boli na dennom poriadku...

Ak by mala táto štúdia priniesť nejaké výsledky tak je to to, že v prvom rade treba zvoliť na riešenie požadovaného problému vhodný LLM nástroj. To sme poznali napríklad na tom, že ChatGPT napríklad nie je schopný spracovať veľké množstvo zdrojových kódov v projekte (už 5+ bol problém), pričom často zabudol na to, čo vytvoril, začal vytvárať tú istú vec znovu, chybné, neúplné... Taktiež napr. častokrát chýbali importy knižníc a podobne.

Ďalším bodom je to, že určite treba niekoho, kto prácu LLM kontroluje. Ak by sme predtým nikdy nerobili v Android Studiu, bolo by veľmi ťažké tento projekt vytvoriť, keďže často bolo potrebné zdrojové kódy vkladať do rôznych adresárov, aktualizovávať gradle závislosti, AndroidManifest...

A v neposlednom rade je dôležitý dobrý a kvalitný prompt engineering, o čom je možné sa presvedčiť v niektorých kapitolách tejto štúdie.