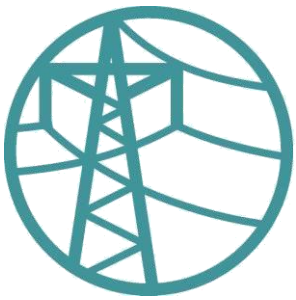

WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Kłubko Paweł

Aplikacja pozwalająca konfigurować zestaw komputerowy

Projekt inżynierski

Opiekun pracy:
dr inż. Antoni Szczepański

Rzeszów, 2023

Spis treści

1. Wstęp

W dzisiejszym dynamicznym środowisku technologicznym, rozwój nowoczesnych rozwiązań informatycznych stanowi kluczowy element postępu. W ramach tej ewolucji, istnieje rosnące zapotrzebowanie na narzędzia umożliwiające łatwą konfigurację i personalizację zestawów komputerowych. W odpowiedzi na tę potrzebę, prezentuję projekt inżynierski o tytule "Aplikacja pozwalająca konfigurować zestaw komputerowy" pisany w języku Kotlin w środowisku Android Studio.

Celem projektu było stworzenie zaawansowanej aplikacji mobilnej, opartej na języku programowania Kotlin, która umożliwi użytkownikom skonfigurowanie i dostosowanie swoich zestawów komputerowych za pomocą intuicyjnego interfejsu graficznego. Zastosowanie technologii mobilnej w połączeniu z potężnym językiem programowania Kotlin pozwoli na stworzenie efektywnego narzędzia, które spełni oczekiwania zarówno doświadczonych entuzjastów komputerowych, jak i osób dopiero rozpoczynających swoją przygodę z tworzeniem własnych konfiguracji sprzętowych.

Projekt zakłada skoncentrowanie się na kilku kluczowych obszarach, takich jak łatwa nawigacja po dostępnych podzespołach, inteligentne propozycje kompatybilnych elementów, czy też możliwość kupienia i wcześniejszego sprawdzenia kompatybilności danej konfiguracji. Ponadto, aplikacja zostanie dostosowana do wymagań systemu Android, co umożliwi użytkownikom korzystanie z niej na różnorodnych urządzeniach mobilnych.

W dalszej części projektu zostaną omówione szczegóły techniczne, cele funkcjonalne oraz planowany zakres funkcji aplikacji. Wierzę, że moje starania przyczynią się do ułatwienia procesu konfiguracji zestawów komputerowych, dostarczając użytkownikom nowoczesne i przyjazne narzędzie, które spełni ich oczekiwania w dziedzinie personalizacji sprzętu komputerowego.

1.1. Opis projektu

Projekt obejmuje rozwiniętą aplikację mobilną napisaną w języku Kotlin przy użyciu środowiska Android Studio. W skład projektu wchodzą różne kluczowe komponenty, których główne funkcje i zastosowania są następujące:

1. **ApiHelper**: Moduł odpowiedzialny za obsługę zapytań API, komunikację z serwerem oraz zarządzanie danymi pobieranymi z zewnętrznych źródeł.
2. **CartActivity**: Aktywność odpowiadająca za interfejs użytkownika związanego z koszykiem zakupów. Wykorzystuje **CartAdapter** do dynamicznego wyświetlania zawartości koszyka.
3. **CartAdapter**: Adapter, który dostarcza mechanizm wiązania danych dla elementów interfejsu użytkownika związanych z koszykiem. Pozwala na efektywne odświeżanie widoków w zależności od zmian w danych.
4. **CartViewModel**: Komponent architektury MVVM, gromadzący dane związane z koszykiem i dostarczający je do interfejsu użytkownika w **CartActivity**.

5. **ComponentsActivity**: Aktywność umożliwiająca przeglądanie i konfigurowanie różnych komponentów komputerowych. Wykorzystuje **ComponentAdapter** do dynamicznego wyświetlania listy komponentów.
6. **MainActivity**: Główna aktywność, od której rozpoczyna się działanie aplikacji. Odpowiada za nawigację pomiędzy różnymi sekcjami aplikacji.
7. **MyApplication**: Klasa reprezentująca główną klasę aplikacji. Może być wykorzystywana do inicjalizacji globalnych zasobów i konfiguracji.
8. **TutorialActivity**: Aktywność zawierająca API Chat GPT umożliwiająca użytkownikowi zadanie określonych pytań dotyczących konkretnych zestawów komputerowych oraz zwracająca odpowiedź na zadane zagadnienia.

W projektowaniu interfejsu graficznego aplikacji wykorzystywane są pliki JSON umieszczone w folderze **assets**, które zawierają dane dotyczące konfiguracji komponentów czy produktów. Ponadto, w folderze **drawable** znajdują się pliki graficzne (**ic_action_shopping_cart**, **ic_baseline_shopping_cart**, **ic_delete**, **ic_launcher_background**, **ic_launcher_foreground**, **logo**), a w folderze **layout** znajdują się pliki XML definiujące układy dla różnych aktywności i elementów interfejsu (**activity_cart**, **activity_components**, **activity_components_for_category**, **activity_main**, **activity_tutorial**, **cart_item**, **component_item**). Pliki graficzne są wykorzystywane do personalizacji interfejsu użytkownika.

2. Główne aktywności aplikacji

W ramach mojego projektu inżynierskiego skupiłem się na stworzeniu serii głównych aktywności, które stanowią rdzeń interakcji użytkownika z moją aplikacją konfiguratora zestawów komputerowych. Te kluczowe elementy dostarczają nie tylko intuicyjnych narzędzi do przeglądania różnych kategorii komponentów, ale również umożliwiają dynamiczną konfigurację oraz zarządzanie zawartością koszyka zakupowego. Przeanalizujmy teraz każdą z tych aktywności, zanurzając się głębiej w ich funkcje oraz rozbudowane możliwości.

MainActivity: Serce Aplikacji

MainActivity pełni rolę głównego punktu wejścia dla użytkowników. To tutaj rozpoczyna się fascynująca podróż po konfiguracji komputerów. W tej aktywności użytkownicy mają możliwość przeglądania różnych komponentów, a także dodawania ich do koszyka zakupowego. Dzięki ComponentAdapter oraz interaktywnym przyciskom, MainActivity oferuje intuicyjne doświadczenie, zachęcając do eksploracji i personalizacji.

ComponentsActivity: Eksploracja Kategorii

ComponentsActivity jest miejscem, gdzie użytkownicy odkrywają bogactwo różnych kategorii komponentów. To tutaj można znaleźć procesory, karty graficzne, płyty główne i wiele innych. Poprzez nawigację między kategoriami, użytkownicy kierowani są do MainActivity, gdzie mogą precyzyjniej konfigurować wybrane podzespoły.

CartActivity: Zarządzanie Zakupami

CartActivity to centralny punkt zarządzania zawartością koszyka zakupowego. Użytkownicy mogą tu dodawać, usuwać i edytować wybrane komponenty. CartAdapter umożliwia dynamiczne odświeżanie widoku w przypadku zmian w koszyku, co zapewnia płynne doświadczenie zarządzania zakupami.

Te główne aktywności stanowią integralną część mojego projektu, tworząc spójne i atrakcyjne środowisko do konfiguracji indywidualnych zestawów komputerowych. Oto kilka kluczowych elementów, które wyróżniają moją aplikację, sprawiając, że proces konfiguracji staje się nie tylko efektywny, ale także satysfakcjonujący dla każdego entuzjasty technologii.

2.1. Aktywność MainActivity.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import android.app.Activity
import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.util.Log
```

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.*
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import org.json.JSONArray
import org.json.JSONException
import android.widget.Button
import android.widget.TextView
import androidx.lifecycle.ViewModelProvider
import java.io.*
import androidx.appcompat.widget.SearchView
import android.widget.ListView

data class Component(
    val name: String,
    val price_usd: String,
    val type: String?,
    val core_clock: String?,
    val boost_clock: String?,
    val core_count: String?,
    val tdp: String?,
): Serializable

class MainActivity : AppCompatActivity() {
    private var isHomePage: Boolean = true
    private lateinit var jsonList: ListView
    private val componentList = ArrayList<Component>()
    private val cartList = ArrayList<Component>()
    private lateinit var cartViewModel: CartViewModel
    private lateinit var searchView: SearchView
    private lateinit var tutorialButton: Button // Dodaj to pole
    private val fullComponentList = ArrayList<Component>()

    private val REQUEST_CART = 1

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        jsonList = findViewById(R.id.json_list)
        val appLogo = findViewById<ImageView>(R.id.appLogo)
        val appName = findViewById<TextView>(R.id.appName)
        tutorialButton = findViewById<Button>(R.id.tutorialButton) // Zainicjalizuj przycisk
        searchView = findViewById(R.id.searchView)

        val selectedCategory = intent.getStringExtra("selectedCategory")
        if (selectedCategory != null) {
            readJsonFile(selectedCategory)
            isHomePage = false
        }

        appLogo.visibility = if (isHomePage) View.VISIBLE else View.GONE
        appName.visibility = if (isHomePage) View.GONE else View.GONE
        tutorialButton.visibility = View.VISIBLE

         // Ukryj lub pokaż pole wyszukiwania w zależności od tego, czy jesteś w kategorii komponentów
        if (isHomePage) {
            searchView.visibility = View.GONE
        } else {
            searchView.visibility = View.VISIBLE
        }

            val componentAdapter = ComponentAdapter(this, R.layout.component_item, componentList)
        jsonList.adapter = componentAdapter

        fullComponentList.addAll(componentList)

        val componentsButton = findViewById<Button>(R.id.componentsButton)
        componentsButton.setOnClickListener {
            val intent = Intent(this, ComponentsActivity::class.java)
            startActivity(intent)
        }
```

```
    val shoppingCartIcon = findViewById<ImageView>(R.id.shoppingCartIcon)
    shoppingCartIcon.setOnClickListener {
        onShoppingCartClick()
    }

    val tutorialButton = findViewById<Button>(R.id.tutorialButton)
    tutorialButton.setOnClickListener {
        val intent = Intent(this, TutorialActivity::class.java)
        startActivity(intent)
    }

    cartViewModel = (application as MyApplication).cartViewModel

    searchView = findViewById(R.id.searchView)
    searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
        override fun onQueryTextSubmit(query: String?): Boolean {
            return false
        }

        override fun onQueryTextChange(newText: String?): Boolean {
            filterComponentList(newText)
            return true
        }
    })
}

private fun filterComponentList(query: String?) {
    if (query.isNullOrBlank()) {
        // Jeśli zapytanie jest puste, przywracamy pełną listę
        (jsonList.adapter as? ComponentAdapter)?.updateList(fullComponentList)
    } else {
        val pattern = query.toRegex(RegexOption.IGNORE_CASE)
        val filteredList = fullComponentList.filter {
            pattern.containsMatchIn(it.name)
        }

        (jsonList.adapter as? ComponentAdapter)?.updateList(filteredList)
```

```
        }
    }

    private fun onShoppingCartClick() {
        val intent = Intent(this, CartActivity::class.java)
        intent.putExtra("cartList", cartViewModel.cartList as Serializable)
        startActivityForResult(intent, REQUEST_CART)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == REQUEST_CART && resultCode == Activity.RESULT_OK) {
            val updatedCartList = data?.getSerializableExtra("updatedCartList") as?
MutableList<Component>
            if (updatedCartList != null) {
                cartViewModel.cartList.clear()
                cartViewModel.cartList.addAll(updatedCartList)
                (jsonList.adapter as? ComponentAdapter)?.notifyDataSetChanged()

                if (updatedCartList.isEmpty()) {
                    val intent = Intent(this, MainActivity::class.java)
                    startActivity(intent)
                }
            } else {
                Log.e("MainActivity", "Updated cart list is null")
            }
        }
    }

    private fun readJsonFile(category: String) {
        try {
            val filename = getCategoryFileName(category)
            val inputStream: InputStream = assets.open(filename)
            val json = BufferedReader(InputStreamReader(inputStream)).use { it.readText() }

            val jsonArray = JSONArray(json)
```

```
            componentList.clear()
            for (i in 0 until jsonArray.length()) {
                val jsonObject = jsonArray.getJSONObject(i)
                val componentName = jsonObject.optString("name", "")
                val componentPrice = jsonObject.optString("price_usd", "")
                val componentType = jsonObject.optString("type", null)
                val componentCoreClock = jsonObject.optString("core_clock", null)
                val componentBoostClock = jsonObject.optString("boost_clock", null)
                val componentCoreCount = jsonObject.optString("core_count", null)
                val componentTdp = jsonObject.optString("tdp", null)

                if (!componentPrice.isNullOrBlank() && componentPrice != "null") {
                    val componentInfo = Component(
                        componentName,
                        componentPrice,
                        componentType,
                        componentCoreClock,
                        componentBoostClock,
                        componentCoreCount,
                        componentTdp
                    )
                    componentList.add(componentInfo)
                }
            }

            (jsonList.adapter as? ComponentAdapter)?.notifyDataSetChanged()
        } catch (e: IOException) {
            e.printStackTrace()
        } catch (e: JSONException) {
            e.printStackTrace()
        }
    }

    private fun getCategoryFileName(category: String): String {
        return when (category) {
            "Komponenty" -> "komponenty.json"
            "Obudowa" -> "case.json"
            "Procesor" -> "cpu.json"
```

```
            "Chłodzenie" -> "cpu-cooler.json"
            "Płyta główna" -> "motherboard.json"
            "Pamięć RAM" -> "memory.json"
            "Karta graficzna" -> "video-card.json"
            "Dysk" -> "external-hard-drive.json"
            "Zasilacz" -> "power-supply.json"
            else -> ""
        }
    }

    inner class ComponentAdapter(
        context: Context,
        private val resource: Int,
        private val components: List<Component>
    ) : ArrayAdapter<Component>(context, resource, components) {

        override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
            val inflater = LayoutInflater.from(context)
            val view = inflater.inflate(resource, parent, false)

            val componentName = view.findViewById<TextView>(R.id.componentName)
            val componentType = view.findViewById<TextView>(R.id.componentType)
            val componentPrice = view.findViewById<TextView>(R.id.componentPrice)

            val addToCartButton = view.findViewById<Button>(R.id.addToCartButton)
            addToCartButton.text = "Dodaj do koszyka"
            addToCartButton.setOnClickListener {
                addToCart(components[position])
            }

            val detailsButton = view.findViewById<Button>(R.id.detailsButton)
            val component = components[position]

            componentName.text = component.name
            componentType.text = component.type
            componentPrice.text = component.price_usd

            detailsButton.text = "Szczegóły"
```

```
            detailsButton.setOnClickListener {
                showComponentDetails(component)
            }

            return view
        }

        fun updateList(newList: List<Component>) {
            clear()
            addAll(newList)
            notifyDataSetChanged()
        }
    }

    private fun addToCart(component: Component) {
        cartViewModel.cartList.add(component)
        (jsonList.adapter as? ComponentAdapter)?.notifyDataSetChanged()
        showAddedToCartMessage(component)
    }

    private fun showComponentDetails(component: Component) {
        val detailsDialog = AlertDialog.Builder(this)
            .setTitle("Szczegóły komponentu")
            .setMessage(buildComponentDetailsMessage(component))
            .setPositiveButton("OK") { dialog, _ ->
                dialog.dismiss()
            }
            .create()

        detailsDialog.show()
    }

    private fun buildComponentDetailsMessage(component: Component): String {
        val message = StringBuilder()

        message.append("Nazwa: ${component.name}\n")
```

```
    if (!component.price_usd.isNullOrBlank() && component.price_usd != "null") {
        message.append("Cena: ${component.price_usd}$\n")
    } else {
        message.append("Cena: Brak dostępnej ceny\n")
    }

    if (!component.type.isNullOrBlank()) {
        message.append("Typ: ${component.type}\n")
    }

    if (!component.core_clock.isNullOrBlank()) {
        message.append("Częstotliwość taktowania: ${component.core_clock}\n")
    }

    if (!component.boost_clock.isNullOrBlank()) {
        message.append("Podkręcone taktowanie: ${component.boost_clock}\n")
    }

    if (!component.core_count.isNullOrBlank()) {
        message.append("Ilość rdzeni: ${component.core_count}\n")
    }
    if (!component.tdp.isNullOrBlank()) {
        message.append("Zasilanie: ${component.tdp}\n")
    }

    return message.toString()
}

private fun showAddedToCartMessage(component: Component) {
    val addedToCartDialog = AlertDialog.Builder(this)
        .setTitle("Dodano do koszyka")
        .setMessage("Komponent ${component.name} został dodany do koszyka.")
        .setPositiveButton("OK") { dialog, _ ->
            dialog.dismiss()
        }
        .create()

    addedToCartDialog.show()
```

```
    }
}
```

MainActivity stanowi serce mojej aplikacji do konfiguracji sprzętu komputerowego. Poniżej przedstawiam szczegółowy opis każdej części klasy:

1. **Deklaracja zmiennych i właściwości:**
    - **isHomePage**: Flaga określająca, czy obecna aktywność to strona główna.
    - **jsonList**: ListView wykorzystywany do wyświetlania listy komponentów.
    - **componentList**: Lista przechowująca obiekty komponentów.
    - **cartList**: Lista przechowująca komponenty dodane do koszyka.
    - **cartViewModel**: Obiekt **CartViewModel** związany z modelem koszyka zakupowego.
    - **searchView**: Element SearchView do filtrowania listy komponentów.
    - **tutorialButton**: Przycisk służący do uruchamiania aktywności z poradnikiem.
    - **fullComponentList**: Lista pełnych danych komponentów, używana do filtrowania.
2. **Metoda onCreate:**
    - Inicjalizacja interfejsu użytkownika z wykorzystaniem **setContentView**.
    - Inicjalizacja elementów interfejsu (np. **jsonList**, **appLogo**, **appName**).
    - Odczytanie kategorii komponentów z przekazanych danych (**selectedCategory**).
    - Ustawienie widoczności elementów w zależności od tego, czy jesteśmy na stronie głównej.
    - Inicjalizacja adaptera (**ComponentAdapter**) i przypisanie go do listy (**jsonList**).
3. **Obsługa przycisków i interakcji:**
    - Obsługa przycisku przechodzenia do kategorii komponentów (**componentsButton**).
    - Obsługa przycisku koszyka zakupowego (**shoppingCartIcon**) i wywołanie metody **onShoppingCartClick**.
    - Obsługa przycisku poradnika (**tutorialButton**) i uruchomienie aktywności **TutorialActivity**.
    - Obsługa wyszukiwania z wykorzystaniem **SearchView**.
4. **Metoda filterComponentList:**
    - Filtracja listy komponentów na podstawie wprowadzonego zapytania.
5. **Metoda onShoppingCartClick:**
    - Obsługa kliknięcia na ikonę koszyka, uruchamia aktywność **CartActivity**.
6. **Metoda onActivityResult:**
    - Obsługa wyników zwracanych po zakończeniu aktywności **CartActivity**.
    - Aktualizacja listy koszyka i wyświetlenie odpowiednich komunikatów.
7. **Metoda readJsonFile:**

- Odczytanie danych komponentów z pliku JSON na podstawie wybranej kategorii.
- Inicjalizacja listy komponentów na podstawie danych z pliku.

8. **Metoda getCategoryFileName:**
    - Mapowanie nazwy kategorii na nazwę pliku JSON.
9. **Klasa ComponentAdapter:**
    - Rozszerzenie ArrayAdapter do obsługi listy komponentów.
    - Nadpisanie metody **getView** dla dostosowania wyglądu elementów listy.
    - Aktualizacja listy w adapterze po wprowadzeniu zmian.
10. **Dodatkowe metody pomocnicze:**

- **addToCart**: Dodanie komponentu do koszyka i wyświetlenie komunikatu.
- **showComponentDetails**: Wyświetlenie szczegółów komponentu w oknie dialogowym.
- **buildComponentDetailsMessage**: Budowanie wiadomości z informacjami o komponencie.
- **showAddedToCartMessage**: Wyświetlenie komunikatu po dodaniu do koszyka.

MainActivity jest kluczowym elementem naszej aplikacji, zapewniającym użytkownikom intuicyjne narzędzia do konfiguracji sprzętu komputerowego oraz efektywne zarządzanie zakupami. Skomplikowana logika interakcji, obsługa zdarzeń oraz interfejsu graficznego sprawiają, że jest to centralna część naszego projektu inżynierskiego.

2.2. Aktywność ApiHelper.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import android.util.Log
import com.google.gson.Gson
import okhttp3.*
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.RequestBody.Companion.toRequestBody
import org.json.JSONArray
import org.json.JSONObject
import java.io.IOException

class ApiHelper {
    companion object {
        private val client = OkHttpClient()
```

```
fun getResponse(question: String, cartList: List<Component>, callback: (String) -> Unit) {
    val apiKey = "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"
    val url = "https://api.openai.com/v1/engines/gpt-3.5-turbo-instruct/completions"

    // Dodaj listę komponentów do kontekstu pytania
    val promptWithCart = buildPrompt(question, cartList)
    Log.d("Prompt", promptWithCart)

    // Utwórz obiekt JSON z danymi do wysłania
    val requestBody = mapOf(
        "prompt" to promptWithCart,
        "max_tokens" to 500,
        "temperature" to 0
    )

    val json = Gson().toJson(requestBody)
    val request = Request.Builder()
        .url(url)
        .addHeader("Content-Type", "application/json")
        .addHeader("Authorization", "Bearer $apiKey")
        .post(json.toRequestBody("application/json".toMediaTypeOrNull()))
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            Log.e("error", "API failed", e)
            callback("API failed: ${e.message}")
        }

        override fun onResponse(call: Call, response: Response) {
            val body = response.body?.string()
            if (body != null) {
                Log.v("data", body)
                try {
                    val jsonObject = JSONObject(body)
                    if (jsonObject.has("choices")) {
```

```
                        val jsonArray: JSONArray = jsonObject.getJSONArray("choices")
                        if (jsonArray.length() > 0 && jsonArray.getJSONObject(0).has("text"))
{
                            val textResult = jsonArray.getJSONObject(0).getString("text")
                            callback(textResult)
                            return
                        }
                    }
                    Log.e("error", "Invalid response format")
                    callback("Invalid response format")
                } catch (e: Exception) {
                    Log.e("error", "Error parsing JSON", e)
                    callback("Error parsing JSON: ${e.message}")
                }
            } else {
                Log.v("data", "empty")
                callback("Empty response from API")
            }
        }
    })
}

private fun buildPrompt(question: String, cartList: List<Component>): String {
    val cartPrompt = cartList.joinToString("\n") { " - ${it.name}: $${it.price_usd}" }
    return "$question\n\nLista komponentów w koszyku:\n$cartPrompt"
}
}
}
```

1. **Companion Object:**
    - Znajdujemy się w bloku **companion object**, który pozwala na utworzenie statycznych funkcji i stałych bez konieczności tworzenia instancji klasy. W tym przypadku **ApiHelper** używa tego mechanizmu do dostarczenia funkcji **getResponse** bez konieczności tworzenia obiektu **ApiHelper**.
2. **Inicjalizacja klienta HTTP:**
    - Klasa posiada prywatne pole **client**, które jest instancją klasy **OkHttpClient**. Jest to popularna biblioteka do obsługi operacji sieciowych w języku Kotlin/Android.
3. **Funkcja getResponse:**

- Jest to główna funkcja klasy, odpowiedzialna za komunikację z API OpenAI. Przyjmuje pytanie, listę komponentów oraz funkcję zwrotną (**callback**) do obsługi rezultatów zapytania.

4. **Zmienne dotyczące API:**
   - W definicji funkcji znajdują się stałe, takie jak klucz API (**apiKey**) i adres URL do usługi OpenAI (**url**).
5. **Budowanie pytania z koszyka:**
   - Wykorzystuje funkcję **buildPrompt**, aby utworzyć pytanie, które uwzględnia zawartość koszyka. Uzyskane pytanie jest następnie zapisywane do logów.
6. **Dane do wysłania:**
   - Tworzy obiekt **requestBody** z danymi, które zostaną wysłane do API. Zawiera prompt, maksymalną liczbę tokenów (500) i temperaturę (0).
7. **Wysłanie zapytania HTTP:**
   - Za pomocą **OkHttpClient** i **Request.Builder** buduje zapytanie HTTP, a następnie wysyła je do usługi OpenAI.
8. **Obsługa odpowiedzi:**
   - Wykorzystuje funkcję **enqueue**, aby asynchronicznie obsłużyć odpowiedź z API.
9. **Obsługa błędów:**
   - W przypadku niepowodzenia zapytania (np. brak dostępu do API) wywołuje funkcję zwrotną z informacją o błędzie.
10. **Odczytanie odpowiedzi:**
    - Odczytuje treść odpowiedzi z API.
11. **Przetworzenie odpowiedzi JSON:**
    - Analizuje odpowiedź JSON, sprawdzając obecność pól "choices" i "text". W przypadku poprawnej odpowiedzi, wywołuje funkcję zwrotną z rezultatem.
12. **Obsługa błędów w odpowiedzi:**
    - W przypadku nieprawidłowego formatu odpowiedzi, loguje błąd i wywołuje funkcję zwrotną z odpowiednim komunikatem.
13. **Obsługa błędów parsowania JSON:**
    - W przypadku problemów z parsowaniem JSON loguje błąd i wywołuje funkcję zwrotną z informacją o błędzie.
14. **Obsługa pustej odpowiedzi:**
    - Gdy odpowiedź z API jest pusta, loguje informację debugową i wywołuje funkcję zwrotną z odpowiednim komunikatem.

2.3. Aktywność CartActivity.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import CartAdapter
import android.app.Activity
import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.gson.Gson
import java.io.Serializable

class CartActivity : AppCompatActivity() {
    private lateinit var cartViewModel: CartViewModel
    private lateinit var cartList: MutableList<Component>
    private var isPaymentCompleted: Boolean = false
    private lateinit var cartRecyclerView: RecyclerView

    // Dodaj EditText i TextView
    private lateinit var questionEditText: EditText
    private lateinit var apiResponseTextView: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_cart)

        // Inicjalizacja ViewModel
        cartViewModel = ViewModelProvider(this).get(CartViewModel::class.java)

        // Odbierz listę komponentów z intencji
```

```
            cartList = intent.getSerializableExtra("cartList") as MutableList<Component>? ?:
mutableListOf()

            // Inicjalizacja RecyclerView
            cartRecyclerView = findViewById(R.id.cartRecyclerView)
            val cartAdapter = CartAdapter(this, cartList) { position ->
                // Obsługa kliknięcia przycisku "Usuń"
                removeItemFromCart(position)
            }
            cartRecyclerView.adapter = cartAdapter
            cartRecyclerView.layoutManager = LinearLayoutManager(this)

            updateTotalPrice(cartList)

            // Inicjalizacja widoków dodanych do layoutu
            questionEditText = findViewById(R.id.questionEditText)
            apiResponseTextView = findViewById(R.id.apiResponseTextView)

            // Dodaj obsługę przycisku sprawdzenia kompatybilności
                                    val checkCompatibilityButton: Button =
findViewById(R.id.checkCompatibilityButton)
            checkCompatibilityButton.setOnClickListener {
                val question = questionEditText.text.toString()

                // Sprawdź kompatybilność z API, przekazując listę komponentów bezpośrednio
                ApiHelper.getResponse(question, cartList) { response ->
                    // Aktualizuj widok w głównym wątku
                    runOnUiThread {
                        apiResponseTextView.text = response
                    }
                }
            }

            // Ustaw początkową listę komponentów w ViewModel
            cartViewModel.cartList.addAll(cartList)

            // Dodaj obsługę przycisku powrotu do komponentów
```

```
                val backToComponentsButton: Button = findViewById(R.id.backToComponentsButton)
                backToComponentsButton.setOnClickListener {
                    val intent = Intent()
                    intent.putExtra("updatedCartList", ArrayList(cartList))
                    setResult(Activity.RESULT_OK, intent)
                    finish()
                }

                // Dodaj obsługę przycisku zapłaty
                val payButton: Button = findViewById(R.id.payButton)
                payButton.setOnClickListener {
                    // Po kliknięciu "Zapłać" zwracamy do MainActivity pusty koszyk
                    isPaymentCompleted = true
                    sendUpdatedCartList(emptyList())

                    // Wyświetlenie komunikatu
                    Toast.makeText(this, "Dziękujemy za zakupy!", Toast.LENGTH_SHORT).show()

                    // Aktualizuj cenę po płatności
                    updateTotalPrice(emptyList())
                }
            }

            private fun sendUpdatedCartList(updatedCartList: List<Component>) {
                // Zapisz aktualizacje w lokalnej zmiennej
                cartList.clear()
                cartList.addAll(updatedCartList)

                val intent = Intent()
                intent.putExtra("updatedCartList", ArrayList(cartList))
                setResult(Activity.RESULT_OK, intent)
                finish()
            }

            private fun updateTotalPrice(cartList: List<Component>) {
                val totalPriceTextView: TextView = findViewById(R.id.totalPrice)
                val totalPrice = calculateTotalPrice(cartList)
```

```
        Log.d("CartActivity", "Total Price: $$totalPrice")
        totalPriceTextView.text = "Total Price: $$totalPrice"
    }

    private fun calculateTotalPrice(cartList: List<Component>): Double {
        var totalPrice = 0.0
        for (component in cartList) {
            component.price_usd.toDoubleOrNull()?.let {
                totalPrice += it
            }
        }
        return totalPrice
    }

    private fun removeItemFromCart(position: Int) {
        if (position >= 0 && position < cartList.size) {
            cartList.removeAt(position)

            // Aktualizuj ViewModel
            cartViewModel.cartList.clear()
            cartViewModel.cartList.addAll(cartList)

            // Aktualizuj cenę po usunięciu elementu
            updateTotalPrice(cartList)

            // Aktualizuj adapter
            cartRecyclerView.adapter?.notifyDataSetChanged()
        }
    }
}
```

1. **Inicjalizacja Zmiennych i Widoków:**
    - Inicjalizacja zmiennych, takich jak **cartViewModel** (odpowiedzialny za przechowywanie danych związanych z koszykiem) oraz **cartList** (aktualna lista komponentów w koszyku).
    - Określenie, czy płatność została zakończona (**isPaymentCompleted**).
    - Inicjalizacja elementów interfejsu użytkownika, takich jak **RecyclerView**, **EditText** i **TextView** do wprowadzania pytań oraz wyświetlania odpowiedzi z API.

2. **Obsługa Intencji:**
    - Odbieranie listy komponentów z poprzedniej aktywności za pomocą intencji.
3. **Inicjalizacja RecyclerView:**
    - Konfiguracja **RecyclerView** za pomocą adaptera (**CartAdapter**), który umożliwia wyświetlanie i usuwanie elementów z koszyka.
4. **Obsługa Przycisków:**
    - **Sprawdzenie Kompatybilności:**
        - Po kliknięciu przycisku "Sprawdź Kompatybilność" pobiera pytanie z **EditText**.
        - Wywołuje funkcję **ApiHelper.getResponse**, aby uzyskać odpowiedź z API w zakresie kompatybilności komponentów.
        - Wyświetla odpowiedź na ekranie w **TextView**.
    - **Powrót do Komponentów:**
        - Przycisk "Powrót do Komponentów" umożliwia użytkownikowi powrót do listy komponentów w poprzedniej aktywności.
        - Przesyła zaktualizowaną listę koszyka do poprzedniej aktywności.
    - **Zapłata:**
        - Po kliknięciu przycisku "Zapłać" ustawia flagę **isPaymentCompleted** na **true**.
        - Wysyła pustą listę do poprzedniej aktywności, sygnalizując, że płatność została zakończona.
        - Wyświetla krótki komunikat Toast informujący o zakończeniu płatności.
        - Aktualizuje cenę na ekranie.
5. **Funkcje Pomocnicze:**
    - **sendUpdatedCartList:**
        - Zapisuje aktualizacje w lokalnej zmiennej **cartList**.
        - Przesyła zaktualizowaną listę koszyka do poprzedniej aktywności za pomocą intencji.
    - **updateTotalPrice:**
        - Oblicza łączną cenę wszystkich komponentów w koszyku.
        - Aktualizuje widok z wyświetlaną ceną.
    - **calculateTotalPrice:**
        - Oblicza łączną cenę komponentów w koszyku na podstawie cen jednostkowych.
    - **removeItemFromCart:**
        - Usuwa komponent z koszyka na podstawie pozycji.
        - Aktualizuje ViewModel, widok i adapter po usunięciu elementu.

W skrócie, **CartActivity** stanowi centralne miejsce do zarządzania koszykiem zakupowym, obsługi płatności oraz komunikacji z API w celu sprawdzenia kompatybilności komponentów. Oferuje również interaktywny interfejs użytkownika, który umożliwia użytkownikowi kontrolę nad zawartością koszyka.

2.4. Aktywność CartAdapter.kt

Kod w języku Kotlin:

```
import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.ImageButton
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.aplikacja_do_konfiguracji_sprztu_komputerowego.Component
import com.example.aplikacja_do_konfiguracji_sprztu_komputerowego.R

class CartAdapter(
    private val context: Context,
    private val cartList: List<Component>,
    private val onRemoveClickListener: (Int) -> Unit
) : RecyclerView.Adapter<CartAdapter.CartViewHolder>() {

    class CartViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val componentName: TextView = itemView.findViewById(R.id.componentName)
        val componentPrice: TextView = itemView.findViewById(R.id.componentPrice)
        val removeButton: ImageButton = itemView.findViewById(R.id.removeButton)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CartViewHolder
{
        val itemView =
            LayoutInflater.from(context).inflate(R.layout.cart_item, parent, false)
        return CartViewHolder(itemView)
    }
```

```
    override fun onBindViewHolder(holder: CartViewHolder, position: Int) {
        val component = cartList[position]

        holder.componentName.text = component.name
        holder.componentPrice.text = component.price_usd

        // Obsługa kliknięcia przycisku usuwania
        holder.removeButton.setOnClickListener {
            if (position >= 0 && position < cartList.size) {
                onRemoveClickListener.invoke(position)
            }
        }
    }

    override fun getItemCount(): Int {
        return cartList.size
    }
}
```

1. **Inicjalizacja ViewHoldera:**
    - **CartAdapter** definiuje klasę wewnętrzną **CartViewHolder**, która dziedziczy po **RecyclerView.ViewHolder**.
    - **CartViewHolder** zawiera trzy elementy interfejsu użytkownika: **componentName** (TextView), **componentPrice** (TextView) oraz **removeButton** (ImageButton).
2. **Metoda onCreateViewHolder:**
    - Tworzy nowy **CartViewHolder**, który reprezentuje poszczególny element interfejsu użytkownika w liście.
    - Wykorzystuje **LayoutInflater** do "nadmuchiwania" widoku z pliku XML (**cart_item.xml**).
3. **Metoda onBindViewHolder:**
    - Ustawia zawartość konkretnego **CartViewHoldera** na podstawie danych z **cartList**.
    - Pobiera komponent z listy na podstawie pozycji.
    - Ustawia nazwę komponentu (**componentName**) i jego cenę (**componentPrice**) na odpowiednich widokach.
    - Dodaje obsługę kliknięcia przycisku usuwania (**removeButton**). Po kliknięciu, wywołuje funkcję **onRemoveClickListener** przekazując aktualną pozycję komponentu.

4. **Metoda getItemCount:**
    - Zwraca liczbę elementów w **cartList**, co odpowiada liczbie elementów w koszyku zakupowym.

W skrócie, **CartAdapter** służy do obsługi danych w **RecyclerView** w **CartActivity**. Tworzy nowe widoki dla elementów koszyka, ustawia ich zawartość na podstawie danych z **cartList** i obsługuje interakcje z użytkownikiem, takie jak kliknięcia przycisków usuwania. Adapter ten jest integralną częścią implementacji koszyka zakupowego w aplikacji do konfiguracji sprzętu komputerowego.

2.5. Aktywność CartViewModel.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import androidx.lifecycle.ViewModel

class CartViewModel : ViewModel() {
   val cartList = mutableListOf<Component>()
}
```

CartViewModel to klasa pochodząca z architektury Android Jetpack, a konkretniej z biblioteki Android Architecture Components, która wspomaga implementację wzorca projektowego Model-View-ViewModel (MVVM).

1. **Dziedziczenie z ViewModel:**
    - **CartViewModel** dziedziczy po klasie **ViewModel** z Android Architecture Components.
    - Dziedziczenie to pozwala na przechowywanie i zarządzanie danymi związanymi z interfejsem użytkownika, takimi jak stany widoków, w sposób, który przetrwa zmiany konfiguracji (np. obrócenie urządzenia).
2. **Zmienna cartList:**
    - **cartList** to zmienna typu **MutableList<Component>**, która przechowuje listę komponentów w koszyku zakupowym.
    - Jest to wspólna lista, którą **CartActivity** i **CartAdapter** korzystają do synchronizacji danych między modelem a widokiem.
3. **Zastosowanie:**
    - **CartViewModel** pełni rolę pośrednika między warstwą modelu (lista komponentów) a warstwą widoku (interfejs użytkownika).

- Przechowuje stan koszyka zakupowego, umożliwiając współdzielenie tych danych między różnymi komponentami aplikacji.

4. **Zarządzanie cyklem życia:**
   - **CartViewModel** automatycznie zarządza swoim cyklem życia w kontekście cyklu życia związanych z nim komponentów (np. **CartActivity**).
   - Dane przechowywane w **CartViewModel** są automatycznie odtwarzane po przerwaniu aktywności, co zapewnia, że nie utracimy stanu koszyka podczas zmiany konfiguracji urządzenia.

W skrócie, **CartViewModel** jest odpowiedzialny za przechowywanie i dostarczanie danych związanych z koszykiem zakupowym, co ułatwia oddzielenie logiki biznesowej od warstwy widoku w ramach wzorca MVVM.

2.6. Aktywność CartViewModel.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity

class ComponentsActivity : AppCompatActivity() {
    companion object {
        const val YOUR_REQUEST_CODE = 123 // Możesz użyć dowolnej liczby całkowitej
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_components)

        val cpuButton = findViewById<Button>(R.id.cpuButton)
        val gpuButton = findViewById<Button>(R.id.gpuButton)
        val caseButton = findViewById<Button>(R.id.caseButton)
        val coolerButton = findViewById<Button>(R.id.coolerButton)
        val motherboardButton = findViewById<Button>(R.id.motherboardButton)
        val memoryButton = findViewById<Button>(R.id.memoryButton)
        val storageButton = findViewById<Button>(R.id.storageButton)
```

```
    val powerSupplyButton = findViewById<Button>(R.id.powerSupplyButton)

    cpuButton.setOnClickListener {
        startCategoryActivity("Procesor")
    }

    gpuButton.setOnClickListener {
        startCategoryActivity("Karta graficzna")
    }

    caseButton.setOnClickListener {
        startCategoryActivity("Obudowa")
    }

    coolerButton.setOnClickListener {
        startCategoryActivity("Chłodzenie")
    }

    motherboardButton.setOnClickListener {
        startCategoryActivity("Płyta główna")
    }

    memoryButton.setOnClickListener {
        startCategoryActivity("Pamięć RAM")
    }

    storageButton.setOnClickListener {
        startCategoryActivity("Dysk")
    }

    powerSupplyButton.setOnClickListener {
        startCategoryActivity("Zasilacz")
    }
}

private fun startCategoryActivity(category: String) {
    val intent = Intent(this, MainActivity::class.java)
    intent.putExtra("selectedCategory", category)
```

```
            startActivityForResult(intent, YOUR_REQUEST_CODE)
        }
        override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
            super.onActivityResult(requestCode, resultCode, data)

                if (requestCode == YOUR_REQUEST_CODE && resultCode ==
Activity.RESULT_OK) {
                    val updatedCartList = data?.getSerializableExtra("updatedCartList") as?
ArrayList<Component>
                updatedCartList?.let {
                    // Zaktualizuj listę komponentów na podstawie odebranej listy
                    // (jeśli potrzebujesz dostępu do tej listy w przyszłości)
                }
            }
        }
    }
```

2.7. Aktywność ComponentsForCategoryActivity.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class ComponentsForCategoryActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_components_for_category)

    }
}
```

Klasa **ComponentsForCategoryActivity** reprezentuje aktywność, która wyświetla komponenty dla konkretnej kategorii.

1. **Dziedziczenie z AppCompatActivity:**

- **ComponentsForCategoryActivity** dziedziczy po klasie **AppCompatActivity**, co oznacza, że jest to aktywność w technologii Android, korzystająca z nowoczesnego modelu projektowania aplikacji zgodnego z Android Jetpack.

2. **Inicjalizacja interfejsu użytkownika:**
   - W metodzie **onCreate** inicjalizowany jest interfejs użytkownika za pomocą widoku (layoutu) zdefiniowanego w pliku XML (**R.layout.activity_components_for_category**).

2.8. Aktywność MyApplication.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import android.app.Application
import androidx.lifecycle.ViewModelProvider

class MyApplication : Application() {
   val cartViewModel: CartViewModel by lazy {
      ViewModelProvider.AndroidViewModelFactory.getInstance(this)
         .create(CartViewModel::class.java)
   }
}
```

Klasa **MyApplication** to klasa rozszerzająca **Application** w Androidzie, a jej głównym celem jest dostarczenie instancji **CartViewModel** na potrzeby aplikacji.

1. **Rozszerzenie klasy Application:**
   - Klasa **MyApplication** jest podklasą klasy **Application**, co oznacza, że jest to specjalna klasa reprezentująca całą aplikację. Jest ona inicjowana, gdy aplikacja jest uruchamiana, a jej istnienie trwa przez cały cykl życia aplikacji.
2. **Inicjalizacja CartViewModel:**
   - Klasa ta zawiera właściwość **cartViewModel**, która jest instancją **CartViewModel**. Jest ona tworzona przy użyciu **lazy**, co oznacza, że jest inicjowana dopiero w momencie pierwszego dostępu do tej właściwości.
3. **Uzyskiwanie instancji CartViewModel:**
   - Wykorzystuje **ViewModelProvider.AndroidViewModelFactory**, aby uzyskać instancję **CartViewModel**. **AndroidViewModelFactory** jest częścią Android Jetpack i jest używana do dostarczania odpowiednich instancji ViewModel dla aplikacji Android.
4. **Żywotność instancji CartViewModel:**

- Ponieważ instancja **CartViewModel** jest uzyskiwana za pomocą **lazy**, będzie ona trwała przez cały cykl życia aplikacji, co jest korzystne, gdy chcemy mieć wspólną instancję ViewModel dostępną dla różnych komponentów aplikacji.

5. **Zastosowanie w aplikacji:**
    - Ta klasa może być używana jako **Application** w pliku AndroidManifest.xml, aby dostarczyć jednej, globalnej instancji **CartViewModel** dla całej aplikacji. Dzięki temu, różne komponenty aplikacji, takie jak różne aktywności, mogą współdzielić tę samą instancję ViewModel i dostęp do wspólnych danych.
6. **Uwagi:**
    - Warto zauważyć, że korzystanie z **ViewModel** w aplikacjach Android jest często praktyką zalecaną, ponieważ pomaga to w organizacji i zarządzaniu danymi między różnymi komponentami, takimi jak aktywności i fragmenty, a także przetrzymywaniu danych podczas zmian konfiguracji.

2.9. Aktywność TutorialActivity.kt

Kod w języku Kotlin:

```
package com.example.aplikacja_do_konfiguracji_sprztu_komputerowego

import android.content.Context
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import okhttp3.*
import org.json.JSONArray
import org.json.JSONObject
import java.io.IOException
import okhttp3.MediaType
import okhttp3.RequestBody
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.RequestBody.Companion.toRequestBody

class TutorialActivity : AppCompatActivity() {
```

```
private val client = OkHttpClient()
private var lastResponse: String? = null
private val sharedPreferencesKey = "lastResponse"

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_tutorial)

    val etQuestion = findViewById<EditText>(R.id.etQuesstion)
    val btnSubmit = findViewById<Button>(R.id.btnSubmit)
    val txtResponse = findViewById<TextView>(R.id.txtResponse)

    // Dodaj przycisk "Komponenty"
    val componentsButton = findViewById<Button>(R.id.componentsButton)
    componentsButton.setOnClickListener {
        navigateToComponentsActivity()
    }

    // Wyświetl ostatnią odpowiedź (jeśli istnieje)
    if (lastResponse != null) {
        txtResponse.text = lastResponse
    }

    // Wczytaj ostatnią odpowiedź z pamięci podręcznej
    lastResponse = loadLastResponse()
    if (lastResponse != null) {
        txtResponse.text = lastResponse
    }

    btnSubmit.setOnClickListener {
        val question = etQuestion.text.toString()
        Toast.makeText(this, question, Toast.LENGTH_SHORT).show()
        getResponse(question) { response ->
            runOnUiThread {
                // Zapisz ostatnią odpowiedź
                lastResponse = response
                txtResponse.text = response
```

```
                    // Zapisz ostatnią odpowiedź w pamięci podręcznej
                    saveLastResponse(response)
                }
            }
        }
    }

    private fun saveLastResponse(response: String) {
        val sharedPreferences = getSharedPreferences(sharedPreferencesKey, Context.MODE_PRIVATE)
        val editor = sharedPreferences.edit()
        editor.putString(sharedPreferencesKey, response)
        editor.apply()
    }

    private fun loadLastResponse(): String? {
        val sharedPreferences = getSharedPreferences(sharedPreferencesKey, Context.MODE_PRIVATE)
        return sharedPreferences.getString(sharedPreferencesKey, null)
    }

    private fun navigateToComponentsActivity() {
        val intent = Intent(this, ComponentsActivity::class.java)
        startActivity(intent)
    }

    private fun getResponse(question: String, callback: (String) -> Unit) {
        val apiKey = "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"
        val url = "https://api.openai.com/v1/engines/gpt-3.5-turbo-instruct/completions"

        val requestBody = """
            {
                "prompt": "$question",
                "max_tokens": 500,
                "temperature": 0
            }
        """.trimIndent()
```

```
    val request = Request.Builder()
        .url(url)
        .addHeader("Content-Type", "application/json")
        .addHeader("Authorization", "Bearer $apiKey")
        .post(requestBody.toRequestBody("application/json".toMediaTypeOrNull()))
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            Log.e("error", "API failed", e)
        }

        override fun onResponse(call: Call, response: Response) {
            val body = response.body?.string()
            if (body != null) {
                Log.v("data", body)
                try {
                    val jsonObject = JSONObject(body)
                    val jsonArray: JSONArray = jsonObject.getJSONArray("choices")
                    val textResult = jsonArray.getJSONObject(0).getString("text")
                    callback(textResult)
                } catch (e: Exception) {
                    Log.e("error", "Error parsing JSON", e)
                }
            } else {
                Log.v("data", "empty")
            }
        }
    })
}
override fun onDestroy() {
    // Wyczyść ostatnią odpowiedź przed zniszczeniem aktywności
    clearLastResponse()
    super.onDestroy()
}

private fun clearLastResponse() {
```

```
            lastResponse = null

            // Wyczyść ostatnią odpowiedź także z pamięci podręcznej
            clearLastResponseFromCache()
        }

        private fun clearLastResponseFromCache() {
                val sharedPreferences = getSharedPreferences(sharedPreferencesKey, Context.MODE_PRIVATE)
            val editor = sharedPreferences.edit()
            editor.remove(sharedPreferencesKey)
            editor.apply()
        }
    }
```

Klasa **TutorialActivity** w mojej aplikacji pełni rolę interaktywnego samouczka, który pozwala użytkownikowi zadawać pytania i otrzymywać odpowiedzi z wykorzystaniem interfejsu programistycznego OpenAI.

1. **Zapisywanie i Wczytywanie Ostatniej Odpowiedzi:**
    - Metody **saveLastResponse** i **loadLastResponse** są odpowiedzialne za zapisywanie ostatniej odpowiedzi do pamięci podręcznej i wczytywanie jej, aby mogła być wyświetlana użytkownikowi po ponownym uruchomieniu aplikacji. Działa to poprzez korzystanie z **SharedPreferences**.
2. **Obsługa Przycisku "Komponenty":**
    - Przycisk "Komponenty" (componentsButton) umożliwia użytkownikowi przejście do aktywności, która wyświetla kategorie komponentów. Obsługa tego przycisku jest realizowana przez metodę **navigateToComponentsActivity**.
3. **Wywoływanie API OpenAI:**
    - Funkcja **getResponse** odpowiada za wywoływanie API OpenAI w celu uzyskania odpowiedzi na zadane pytanie. Wykorzystuje ona klienta OkHttp do wysłania żądania POST do odpowiedniego endpointu API OpenAI.
4. **Aktualizacja Interfejsu Użytkownika:**
    - Aktualizacje interfejsu użytkownika, takie jak wyświetlanie ostatniej odpowiedzi i prezentowanie nowej odpowiedzi, są obsługiwane w funkcji **runOnUiThread**.
5. **Oczekiwanie na Odpowiedź Z API (Callback):**
    - API OpenAI jest wywoływane asynchronicznie, a funkcja **getResponse** przyjmuje funkcję zwrotną (callback), która zostanie wywołana po otrzymaniu odpowiedzi od API. W tej funkcji zwrotnej następuje aktualizacja interfejsu użytkownika.

6. **Czyszczenie Pamięci Przed Zniszczeniem Aktywności:**
    - Przed zniszczeniem aktywności (w metodzie **onDestroy**), ostatnia odpowiedź jest wyczyszczona, a także usuwana jest zapisana w pamięci podręcznej.
7. **Użycie Toast:**
    - Wykorzystanie **Toast** do wyświetlania krótkich komunikatów dla użytkownika, takich jak wyświetlanie treści pytania po naciśnięciu przycisku "Submit".
8. **Inicjalizacja Elementów Interfejsu Użytkownika:**
    - Inicjalizacja widoków, takich jak **EditText**, **Button** i **TextView**, które umożliwiają użytkownikowi wprowadzanie pytań, przesyłanie ich do API i wyświetlanie odpowiedzi.
9. **Nawigacja do Innej Aktywności:**
    - Przejście do aktywności wyświetlającej kategorie komponentów po naciśnięciu przycisku "Komponenty".

W ogólnym kontekście, **TutorialActivity** dostarcza interaktywnego środowiska dla użytkownika, w którym może zadawać pytania za pomocą interfejsu API OpenAI i otrzymywać odpowiedzi. Ostatnie odpowiedzi są przechowywane w pamięci podręcznej, co pozwala na wyświetlanie ich po ponownym uruchomieniu aplikacji.

3. Layout aplikacji

Struktura interfejsu aplikacji została starannie zaprojektowana, skupiając się na intuicyjnej nawigacji i estetycznym doświadczeniu użytkownika. Zastosowano przejrzysty układ, w którym główne komponenty interfejsu są łatwo dostępne, co umożliwia użytkownikowi płynne poruszanie się między funkcjonalnościami aplikacji.

Na ekranie głównym wita nas logo aplikacji oraz nazwa, natomiast centralnym elementem jest lista komponentów w formie dynamicznej listy, zintegrowanej z możliwością dynamicznego wyszukiwania. Dzięki temu użytkownik może łatwo przeglądać i znajdować interesujące go elementy.

Dodatkowo, interfejs dostarcza ikonę koszyka, która stanowi centralne miejsce gromadzenia wybranych komponentów. Przejście do koszyka jest zintegrowane z intuicyjnym przyciskiem, zapewniając pełną kontrolę nad aktualnie wybranymi elementami.

Przyciski nawigacyjne do kategorii komponentów są wyraźnie widoczne, ułatwiając użytkownikowi szybkie przechodzenie między różnymi rodzajami sprzętu komputerowego. To wszystko tworzy spójny i funkcjonalny layout, który sprzyja efektywnej konfiguracji sprzętu.

3.1. Layout activity_cart.xml

Kod w XML:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".CartActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/cartRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <TextView
        android:id="@+id/totalPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
    android:text="Total Price: $0.00"
    android:textSize="18sp"
    android:textStyle="bold"
    android:padding="16dp"/>

<Button
    android:id="@+id/backToComponentsButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Powrót do komponentów"
    android:layout_gravity="center"
    android:layout_marginTop="16dp"/>

<Button
    android:id="@+id/payButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Zapłać"
    android:layout_below="@id/backToComponentsButton"
    android:layout_marginTop="16dp"/>
<EditText
    android:id="@+id/questionEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Zadaj pytanie API" />

<Button
    android:id="@+id/checkCompatibilityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sprawdź"
    android:layout_marginTop="16dp" />

<TextView
    android:id="@+id/apiResponseTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Odpowiedź z API"
```

```
    />
</LinearLayout>
```

Activity Cart, reprezentowane przez plik layoutu XML, zapewnia interfejs użytkownika do zarządzania zawartością koszyka zakupowego w aplikacji konfiguracji sprzętu komputerowego.

1. **RecyclerView (@+id/cartRecyclerView):**
    - Jest to komponent RecyclerView, który dynamicznie wyświetla listę komponentów w koszyku.
    - Skonfigurowany jest z użyciem adaptera **CartAdapter**, który dostarcza dane do wyświetlenia.
    - Ustawiono parametry takie jak **layout_width**, **layout_height**, **layout_weight**, aby poprawnie dostosować się do interfejsu.
2. **TextView (@+id/totalPrice):**
    - Wyświetla łączną cenę wszystkich komponentów znajdujących się w koszyku.
    - Zastosowano formatowanie tekstu, ustawiając rozmiar, styl i odstępy.
3. **Button (@+id/backToComponentsButton):**
    - Przycisk umożliwiający powrót do ekranu wyboru komponentów.
    - Posiada tekst "Powrót do komponentów" oraz jest wycentrowany.
4. **Button (@+id/payButton):**
    - Przycisk służący do zakończenia procesu zakupów i przechodzenia do płatności.
    - Posiada tekst "Zapłać".
5. **EditText (@+id/questionEditText):**
    - Pole do wprowadzania pytania, które zostanie przekazane do zapytania API.
    - Używane w celu sprawdzenia kompatybilności wybranych komponentów.
6. **Button (@+id/checkCompatibilityButton):**
    - Przycisk, który inicjuje sprawdzenie kompatybilności wybranych komponentów poprzez zapytanie do API.
    - Wyświetla tekst "Sprawdź".
7. **TextView (@+id/apiResponseTextView):**
    - Wyświetla odpowiedź z API w związku ze sprawdzeniem kompatybilności komponentów.
    - Domyślnie zawiera tekst "Odpowiedź z API".

Interfejs ten umożliwia użytkownikowi zarządzanie koszykiem zakupowym, kontrolowanie komponentów, zadawanie pytań do API i sprawdzanie kompatybilności.

3.2. Layout activity_components.xml

Kod w XML:

```
<!-- activity_components.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ComponentsActivity">

    <androidx.appcompat.widget.SearchView
        android:id="@+id/searchView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:queryHint="Szukaj komponentu..."
        android:layout_marginTop="16dp"
        android:layout_marginHorizontal="16dp"
        android:iconifiedByDefault="false"
        android:focusable="false"
        android:focusableInTouchMode="true" />

    <Button
        android:id="@+id/cpuButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Procesor"
        android:layout_marginBottom="8dp"/>

    <Button
        android:id="@+id/gpuButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Karta graficzna"
        android:layout_marginBottom="8dp"/>

    <Button
        android:id="@+id/caseButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
    android:text="Obudowa"
    android:layout_marginBottom="8dp"/>

<Button
    android:id="@+id/coolerButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Chłodzenie"
    android:layout_marginBottom="8dp"/>

<Button
    android:id="@+id/motherboardButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Płyta główna"
    android:layout_marginBottom="8dp"/>

<Button
    android:id="@+id/memoryButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Pamięć RAM"
    android:layout_marginBottom="8dp"/>

<Button
    android:id="@+id/storageButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Dysk"
    android:layout_marginBottom="8dp"/>

<Button
    android:id="@+id/powerSupplyButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Zasilacz"
    android:layout_marginBottom="8dp"/>
<ListView
```

```
        android:id="@+id/components_list"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">
    </ListView>

    <!-- Dodaj inne przyciski według potrzeb -->

</LinearLayout>
```

Activity Components, reprezentowane przez plik layoutu XML, dostarcza interfejs użytkownika do wyboru kategorii komponentów do konfiguracji sprzętu komputerowego.

1. **SearchView (@+id/searchView):**
    - Komponent SearchView, który umożliwia użytkownikowi wyszukiwanie komponentów według nazwy.
    - Wyposażony w podpowiedzi (**queryHint**) i odpowiednie marginesy.
2. **Button (@+id/cpuButton):**
    - Przycisk reprezentujący kategorię procesorów.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych procesorów.
3. **Button (@+id/gpuButton):**
    - Przycisk reprezentujący kategorię kart graficznych.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych kart graficznych.
4. **Button (@+id/caseButton):**
    - Przycisk reprezentujący kategorię obudów.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych obudów.
5. **Button (@+id/coolerButton):**
    - Przycisk reprezentujący kategorię chłodzeń.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych chłodzeń.
6. **Button (@+id/motherboardButton):**
    - Przycisk reprezentujący kategorię płyt głównych.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych płyt głównych.
7. **Button (@+id/memoryButton):**
    - Przycisk reprezentujący kategorię pamięci RAM.

- Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych pamięci RAM.

8. **Button (@+id/storageButton):**
    - Przycisk reprezentujący kategorię dysków.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych dysków.
9. **Button (@+id/powerSupplyButton):**
    - Przycisk reprezentujący kategorię zasilaczy.
    - Po naciśnięciu przycisku aplikacja przechodzi do wyświetlania dostępnych zasilaczy.
10. **ListView (@+id/components_list):**
    - Komponent ListView, który może być wykorzystany do wyświetlania listy komponentów (np. po wyborze kategorii).
    - Skonfigurowany z użyciem adaptera, aby dostarczać dane do wyświetlenia.

Interfejs ten umożliwia użytkownikowi przeglądanie dostępnych kategorii komponentów, wyszukiwanie, a także przechodzić do wybierania konkretnych produktów w ramach każdej kategorii.

3.3. Layout activity_components_for_category.xml

Kod w XML:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ComponentsForCategoryActivity">

</androidx.constraintlayout.widget.ConstraintLayout>
```

Plik layoutu XML dla **ComponentsForCategoryActivity** definiuje strukturę widoku za pomocą ConstraintLayout, ale nie zawiera jeszcze żadnych widocznych komponentów interfejsu użytkownika. Obecnie jest to pusty layout, a więc nie zawiera konkretnego opisu, co się w nim dzieje.

3.4. Layout activity_main.xml

Kod w XML:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".MainActivity">

<androidx.appcompat.widget.SearchView
    android:id="@+id/searchView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:queryHint="Szukaj komponentu..."
    android:layout_marginTop="16dp"
    android:layout_marginHorizontal="16dp"
    android:iconifiedByDefault="false"
    android:focusable="false"
    android:focusableInTouchMode="true" />

<ImageView
    android:id="@+id/appLogo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/logo"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"/>

<TextView
    android:id="@+id/appName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Aplikacja do konfiguracji sprzętu komputerowego"
    android:textSize="18sp"
    android:textColor="@android:color/black"
    android:layout_gravity="end"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:visibility="gone"/>
```

```
<ListView
    android:id="@+id/json_list"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1">
</ListView>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="end"
    android:layout_marginEnd="16dp"
    android:layout_marginTop="16dp">

    <ImageView
        android:id="@+id/shoppingCartIcon"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:src="@drawable/ic_baseline_shopping_cart"
        android:layout_gravity="end"
        android:clickable="true"
        android:onClick="onShoppingCartClick"/>
</LinearLayout>

<Button
    android:id="@+id/componentsButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Komponenty"
    android:layout_gravity="center_horizontal|bottom"
    android:layout_marginTop="16dp"
    android:layout_marginBottom="8dp"/> <!-- Zmienione odstępy -->

<Button
    android:id="@+id/tutorialButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
        android:text="Poradnik"
        android:layout_gravity="center_horizontal|bottom"
        android:layout_marginTop="8dp" />
    </LinearLayout>
```

Plik layoutu XML dla **MainActivity** definiuje interfejs użytkownika w formie pionowego układu **LinearLayout.**

1. **SearchView:** To komponent służący do wprowadzania tekstu w celu wyszukiwania. Ma nadany identyfikator **searchView** i jest umieszczony na górze ekranu.
2. **ImageView (appLogo):** Wyświetla logo aplikacji. Jest umieszczone w centrum ekranu, a jego źródło obrazu (**src**) wskazuje na zasób o nazwie "logo". Jest to obrazek, który identyfikuje twoją aplikację.
3. **TextView (appName):** Wyświetla nazwę aplikacji. Tekst ma rozmiar 18sp, kolor czarny, a widoczność ustawiona jest na "gone" (nie widoczny). Może być używany do dynamicznego wyświetlania lub ukrywania nazwy aplikacji w zależności od potrzeb.
4. **ListView (json_list):** To komponent listy, który może wyświetlać dane w formie listy. W tym przypadku, jego identyfikator to **json_list**. Jest ustawiony na zajmowanie dostępnego miejsca, ale dostosowuje się do przypisanej mu wagi (**layout_weight**).
5. **LinearLayout (shoppingCartIcon):** Jest to układ poziomy zawierający ikonę koszyka (**shoppingCartIcon**). Ikona jest klikalna, co sugeruje, że po kliknięciu uruchamiana jest metoda **onShoppingCartClick**.
6. **Button (componentsButton):** Przycisk, który po naciśnięciu uruchamia aktywność związaną z wyborem komponentów. Jest ustawiony na zajmowanie dostępnego miejsca z odstępem na górze i na dole.
7. **Button (tutorialButton):** Przycisk, który po naciśnięciu uruchamia aktywność związana z poradnikiem. Jest ustawiony na zajmowanie dostępnego miejsca z odstępem na górze.

Ten plik layoutu definiuje układ interfejsu użytkownika w **MainActivity**, obejmujący elementy takie jak wyszukiwanie, logo aplikacji, listę, ikonę koszyka, przyciski i inne.

3.5. Layout activity_tutorial.xml

Kod w XML:

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

```
tools:context=".TutorialActivity"
android:padding="20dp">

<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content">

   <EditText
      android:id="@+id/etQuesstion"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Czego chcesz się dowiedzieć?"
      android:textSize="20sp"
      android:layout_alignParentTop="true"/>

   <Button
      android:id="@+id/btnSubmit"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Zatwierdź"
      android:layout_below="@+id/etQuesstion"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="20dp"
      android:layout_marginBottom="20dp"/>

   <TextView
      android:id="@+id/txtResponse"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Najelpsze rozwiązanie dla twojego problemu pojawi się tutaj!"
      android:textSize="20sp"
      android:layout_below="@+id/btnSubmit"
      android:layout_marginTop="20dp"
      android:layout_marginBottom="20dp"/>

   <Button
      android:id="@+id/componentsButton"
      android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Komponenty"
            android:layout_below="@+id/txtResponse"
            android:layout_alignParentBottom="true"/>
    </RelativeLayout>
</ScrollView>
```

Plik layoutu XML dla **TutorialActivity** definiuje interfejs użytkownika w formie pionowego układu **ScrollView.**

1. **ScrollView:** Kontener, który umożliwia przewijanie treści, gdy zawartość nie mieści się na ekranie. Zawiera jeden element podrzędny, tj. **RelativeLayout**.
2. **RelativeLayout:** Układ, w którym elementy są rozmieszczane względem siebie lub względem rodzica. Zawiera następujące elementy:
    - **EditText (etQuesstion):** Pole do wprowadzania tekstu, w którym użytkownik może wpisywać pytanie. Ma ustawiony hint, a jego tekst ma rozmiar 20sp. Jest ustawione na górze ekranu (**layout_alignParentTop**).
    - **Button (btnSubmit):** Przycisk do zatwierdzania wprowadzonego pytania. Jest umieszczony poniżej pola tekstowego (**layout_below**) i wycentrowany względem szerokości ekranu.
    - **TextView (txtResponse):** Pole tekstowe, w którym pojawi się odpowiedź na pytanie użytkownika. Ma ustawiony hint oraz rozmiar tekstu 20sp. Jest umieszczone poniżej przycisku **btnSubmit** i może być przewijane, jeśli treść jest zbyt długa.
    - **Button (componentsButton):** Przycisk, który po naciśnięciu uruchamia aktywność związaną z wyborem komponentów. Jest umieszczony na dole ekranu (**layout_alignParentBottom**).

Ten plik layoutu definiuje interfejs użytkownika dla **TutorialActivity**, który umożliwia użytkownikowi wprowadzanie pytania, zatwierdzanie go, wyświetlanie odpowiedzi oraz nawigację do aktywności wyboru komponentów.

Sygnatura:

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2023
Wydział Elektrotechniki i Informatyki

STRESZCZENIE PROJEKTU INŻYNIERSKIEGO

Aplikacja pozwalająca konfigurować zestaw komputerowy

Autor: Paweł Kłubko, nr albumu: EF-DI-167803 Opiekun: dr inż.
Antoni Szczepański
Słowa kluczowe: Konfiguracja komputerowa, Sklep komputerowy, OpenAI, Komponenty komputerowe, Kompatybilność sprzętu

Aplikacja umożliwia konfigurację sprzętu komputerowego, pozwalając użytkownikowi skompletować listę potrzebnych komponentów. Zintegrowany sklep pozwala na przeglądanie i wybieranie elementów. Wykorzystuje także OpenAI do sprawdzania kompatybilności i udziela porad.

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2023
Faculty of Electrical and Computer Engineering

DIPLOMA THESIS (BS) ABSTRACT

An application that allows you to configure your computer set

Author: Paweł Kłubko, code: EF-DI -167803 Supervisor: dr inż.
Antoni Szczepański
Key words: Computer Configuration, Computer Store, OpenAI, Computer Components, Hardware Compatibility

The application facilitates computer hardware configuration, allowing users to assemble a list of required components. An integrated store enables browsing and selecting items. It also utilizes OpenAI to check compatibility and provide advice.