

# Type Racer

## Aplikacje mobilne dla systemu Android

Artur Bednarczyk, Dominika Jurczyk, Damian Fikier

Politechnika Śląska

Wydział Matematyki Stosowanej

Informatyka, semestr V

7 stycznia 2019



**Politechnika  
Śląska**

# Spis treści

<b>1</b>	<b>Zespół</b>	<b>3</b>
<b>2</b>	<b>Opis projektu</b>	<b>4</b>
2.1	Opis . . . . .	4
2.2	Projekt UI . . . . .	4
2.3	Funkcjonalności . . . . .	5
2.3.1	Gra . . . . .	5
2.3.2	Lista wyników . . . . .	5
2.3.3	Zgłaszanie własnego wyniku . . . . .	5
2.3.4	Instrukcja i opis . . . . .	5
<b>3</b>	<b>Technologie, narzędzia</b>	<b>5</b>
<b>4</b>	<b>Implementacja</b>	<b>6</b>
4.1	Podział projektu na pliki . . . . .	6
4.2	Architektura . . . . .	7
4.3	Schemat Modelu Obiektowego . . . . .	9
4.4	API . . . . .	9

# 1 Zespół

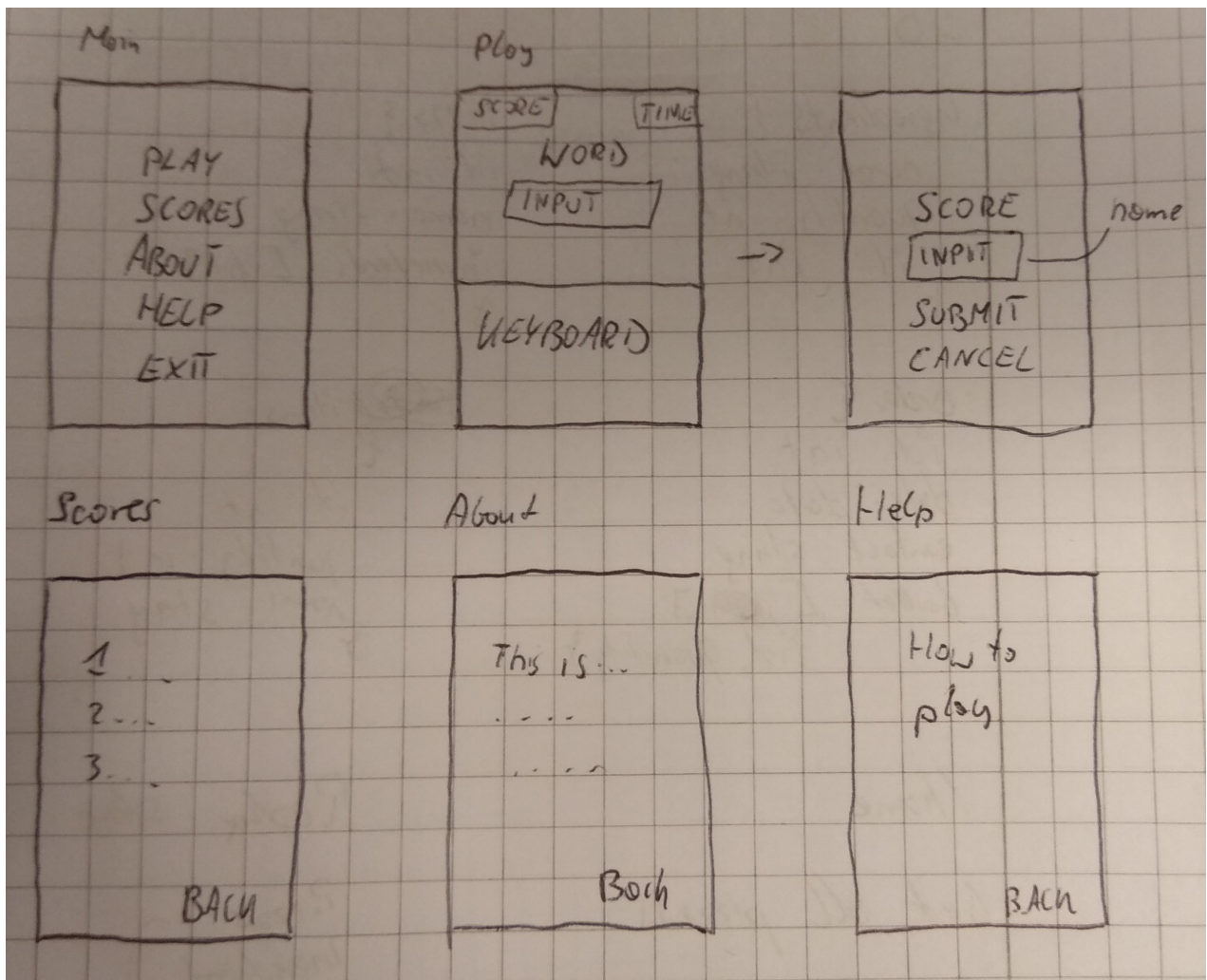
- Bednarczyk Artur
  - Projekt aplikacji - UI i funkcjonalności
  - Serwer z Node.js
  - Struktura aplikacji - MVP
  - Fragmenty i zarządzanie nimi
  - RecyclerViewAdapter dla listy wyników
  - Model danych
  - "Dialog" do przesyłania wyniku
  - Logika gry
  - Dokumentacja
- Jurczyk Dominika
  -
- Fikier Damian
  - Obsługa API
  - Obsługa JSON

## 2 Opis projektu

### 2.1 Opis

Gra „Type-Racer”, która polega na wpisywaniu słów pojawiających się na ekranie. Aby słowo zostało zaliczone, musi być wpisane w pełni poprawnie. Po zaliczeniu słowa gracz otrzymuje punkt i pojawia się kolejne słowo. Rozgrywka trwa określony czas. Po zakończeniu gracz ma możliwość przesłania swojego wyniku na serwer, gdzie jest przechowywana lista najlepszych wyników, którą będzie można zobaczyć w aplikacji. Aplikacja będzie posiadała również instrukcję i opis.

### 2.2 Projekt UI



## **2.3 Funkcjonalności**

### **2.3.1 Gra**

Wpisywanie jak najszybciej wyświetlonego słowa, poprawne wpisanie słowa gwarantuje punkt oraz wyświetlenie kolejnego słowa. Im więcej słów zostanie wpisanych poprawnie, tym więcej punktów uzyska gracz.

### **2.3.2 Lista wyników**

Gracz ma możliwość zobaczenia listy najlepszych przesłanych wyników.

### **2.3.3 Zgłaszanie własnego wyniku**

Po zakończeniu rozgrywki gracz może przesłać swój wynik na serwer.

### **2.3.4 Instrukcja i opis**

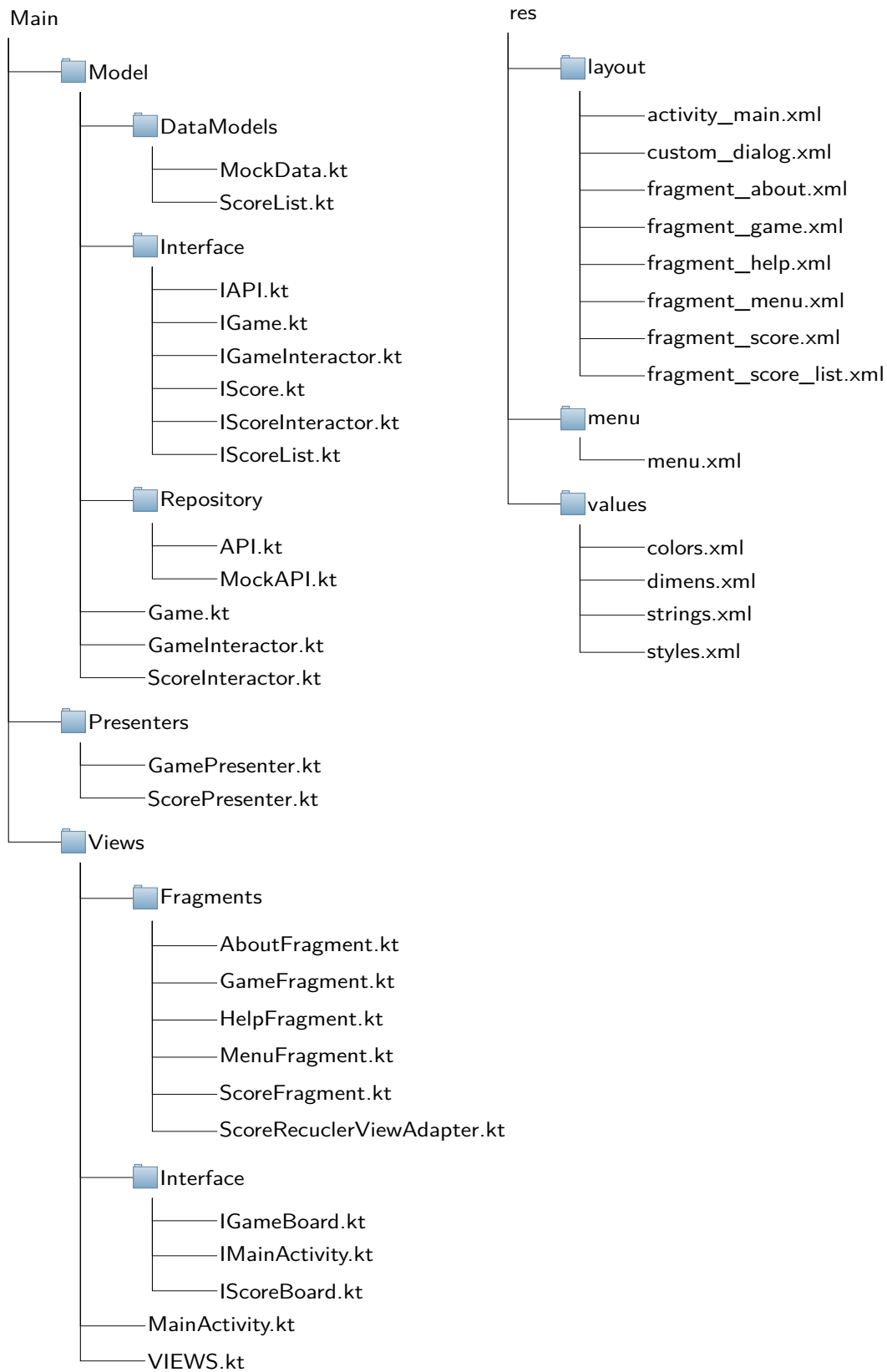
Gra zawiera instrukcję oraz opis.

## **3 Technologie, narzędzia**

- Android Studio - Środowisko programistyczne.
- GitHub - Repozytorium do przechowywania wersji online.
- Heroku - platforma, która przechowuje nasz serwer w chmurze.
- mLab - baza danych na listę wyników
- Kotlin - aplikacja na platformę Android
- MongoDB - baza danych
- Node.js - serwer

## 4 Implementacja

### 4.1 Podział projektu na pliki



## 4.2 Architektura

Aplikacja składa się z jednej aktywności, która zawiera fragmenty. Dzięki implementacji odpowiedniego interfejsu, fragmenty mogą komunikować się z aktywnością, co jest wykorzystywane do przełączania się między fragmentami. Fragment z głównym menu, po kliknięciu odpowiedniego przycisku wysyła informację o tym do aktywności, która podmieni fragment.

---

```
// MenuFragment.kt
private var listenerMenu: OnMenuFragmentInteractionListener? = null
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_menu, container,
            false)
        view.playButton.setOnClickListener {
            listenerMenu?.onMenuFragmentInteraction(VIEWS.GAME)
        }
    }
    interface OnMenuFragmentInteractionListener {
        fun onMenuFragmentInteraction(s: VIEWS)
    }
```

---

Listing 1: Interfejs oraz wywołanie akcji w fragmencie

---

```
// MainActivity.kt
    override fun onMenuFragmentInteraction(s: VIEWS) {
        when (s) {
            VIEWS.MENU -> changeFragment(menuFragment)
            VIEWS.GAME -> changeFragment(gameFragment)
            VIEWS.SCORE -> changeFragment(scoreFragment)
            VIEWS.HELP -> changeFragment(helpFragment)
            VIEWS.ABOUT -> changeFragment(aboutFragment)
            VIEWS.EXIT -> exitGame()
        }
    }
```

---

Listing 2: Implementacja w aktywności

Zastosowany wzorzec Model-View-Presenter pozwolił na oddzielenie logiki od widoku. Aby połączenie było cały czas aktywne ustanawiamy je w metodzie onCreateView danego fragmentu. Konstruktor prezentera wymaga również modelu jaki chcemy stosować. Przykład:

---

```
\\ GameFragment.kt
lateinit var presenter: GamePresenter
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_game, container,
            false)
        presenter = GamePresenter(this, GameInteractor())
        presenter.getWord()
```

---

```
        return view
    }
```

---

Listing 3: Połączenie fragmentu z prezydentem

Przykładowa implementacja prezentera:

```
// GamePresenter.kt
class GamePresenter(val view: IGameBoard, val interactor: IGameInteractor){
    fun getWord() {
        view.wordInput.text = interactor.getWord()
    }
}
```

---

Listing 4: Prezenter

Prezenter komunikując się z modelem, korzysta z "Interactor", który odpowiada za interakcje z danymi.

```
// GameInteractor.kt
class GameInteractor : IGameInteractor {
    val API = MockAPI
    override fun getWord(): String {
        return API.getWord()
    }
}
```

---

Listing 5: Interactor

Dane wykorzystywane w aplikacji pobierane są z serwera za pomocą "Repository", które zawiera metody odpowiedzialne za wykonywanie zapytań do zewnętrznego serwera. W ramach testowania utworzono fałszywe API

```
// MockAPI.kt
object MockAPI: IAPI {
    override fun getWord(): String {
        return "randomWORDtest"
    }
}
```

---

Listing 6: Repository

Model danych:

```
// ScoreList.kt
class ScoreList: IScoreList {
    override val SCORES: MutableList<Score> = ArrayList()

    override fun addScore(score: Score){
        SCORES.add(score)
    }
}
```



```
data class Score(override val score: Int, override val nick: String) :  
    IScore  
}
```

---

Listing 7: Model Danych

### 4.3 Schemat Modelu Obiektowego

Rysunek 1: Tu będzie schemat

### 4.4 API

Adres serwera: <http://simple-type-racer.herokuapp.com/>

- Test połączenia
  - URL: /server
  - metoda: GET
  - parametry url: brak
  - odpowiedz: JSON
  - przykładowa odpowiedź:

```
{  
  "connection": true  
}
```

- 1 słowo
  - URL: /server/getWord
  - metoda: GET
  - parametry url: brak
  - odpowiedz: STRING
  - przykładowa odpowiedź:

```
"word"
```

- 5 słów
  - URL: /server/getWord
  - metoda: GET
  - parametry url: brak
  - odpowiedz: tablica 5 elementów typu: STRING

```
["quae", "quia", "autem", "facere", "officiis"]
```

- zgłaszanie wyniku

- URL: /server/result
- metoda: POST
- parametry url: brak
- parametry w ciele: nickname=[String] oraz score=[Number]
- przykładowe ciało zapytania:

```
{  
  "nickname": "name",  
  "score": 23  
}
```

- odpowiedz: JSON
- przykładowa odpowiedź:

```
{  
  "success": true,  
  "info": {  
    "_id": "5c27d0d7dd97760015a5391b",  
    "nickname": "Isur",  
    "score": 11,  
    "__v": 0  
  }  
}
```

- top 10

- URL: /server/top10
- metoda: GET
- parametry url: brak
- odpowiedz: JSON
- przykładowa odpowiedź:

```
[  
  {  
    "_id": "5c1ff3415b36030015bd61c4",  
    "nickname": "User1",  
    "score": 38,  
    "__v": 0  
  },  
  {
```

```
    "_id": "5c27d0d7dd97760015a5391b",  
    "nickname": "User2",  
    "score": 11,  
    "--v": 0  
  },  
]
```