Adrian Piniek 144226
Dawid Królak 145383

Aplikacja dla osób uprawiających jogging

7 czerwca 2022

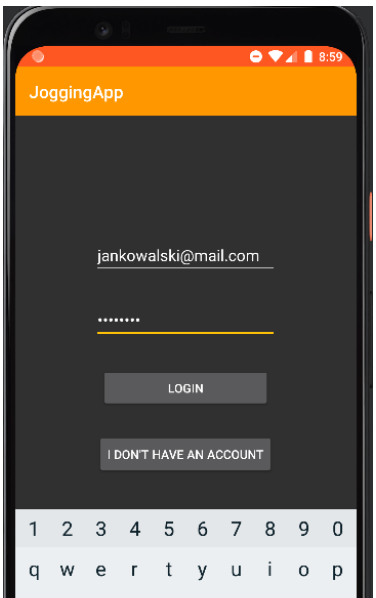
OPIS APLIKACJI

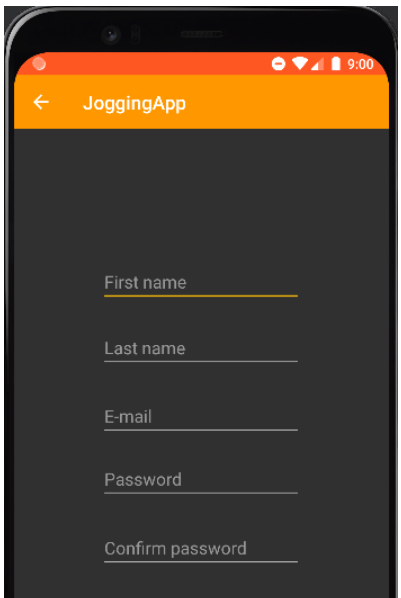
Założeniem projektu było stworzenie aplikacji przeznaczonej dla sportowców szukających tras pieszych / joggingowych / rowerowych w swojej okolicy. Aplikacja pokazuje listy tras z podziałem na łatwe, średnie i trudne oraz pozwala zobaczyć ich szczegóły, takie jak długość trasy, jej położenie, oraz opis. W widoku szczegółów trasy funkcjonuje stoper, za którego pomocą można zapisać osiągnięte na trasie wyniki do bazy danych i porównać swój ostatnio osiągnięty czas z osobistym rekordem. Aplikacja działa zarówno na smartfonach jak i tabletach, w każdym momencie można także dokonać obrotu urządzenia, bez utraty wyświetlanych danych.

OPIS DZIAŁANIA

Logowanie

Na ekranie startowym aplikacji użytkownik jest witany formularzem logowania do aplikacji. Jeśli nie posiada konta, może takowe założyć poprzez panel rejestracji znajdujący się w osobnej aktywności, uruchamianej przez naciśnięcie przycisku ‘I don’t have an account’. Z ekranu rejestracji można powrócić do ekranu logowania za pomocą przycisku strzałki znajdującego się





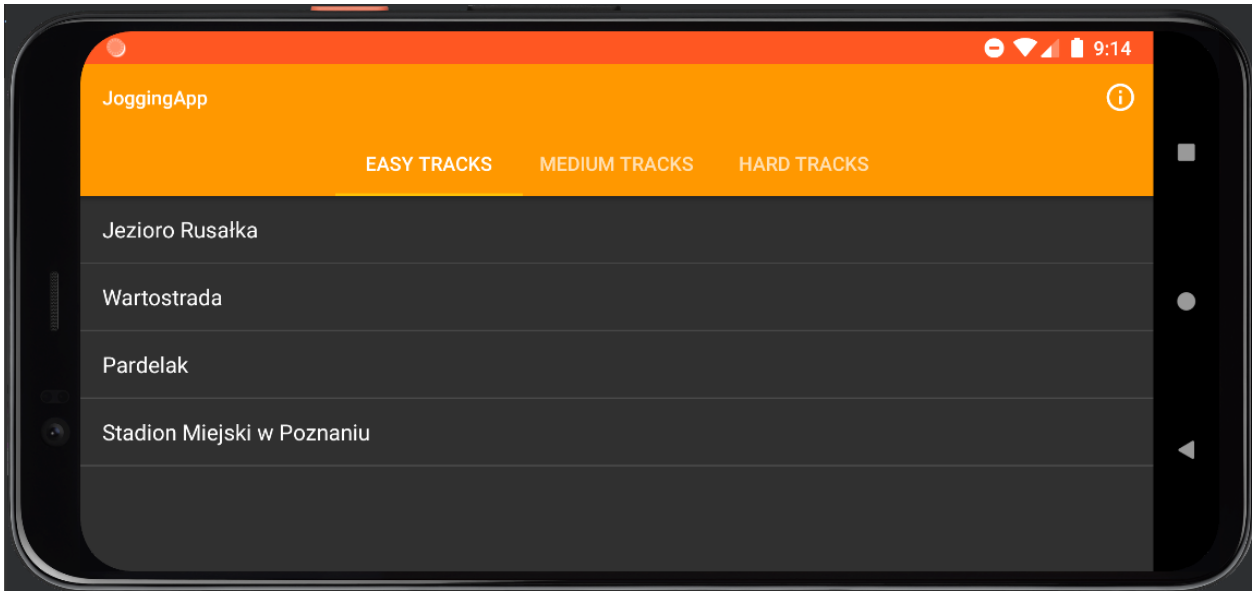
w toolbarze. Po pomyślnej rejestracji, użytkownik może się zalogować i przejść do właściwej aplikacji.

```
private fun login() {
    val editTextMail = findViewById<EditText>(R.id.editTextMail)
    val editTextPassword = findViewById<EditText>(R.id.editTextPassword)
    val email = editTextMail.text.toString()
    val password = editTextPassword.text.toString()

    if(email.isEmpty() || password.isEmpty()) {
        toast( text: "All the fields have to be filled")
    }
    else {
        val user = User(email, password)
        val call = serviceGenerator.login(user)

        call.enqueue(object : Callback<User> {
            override fun onResponse(call: Call<User>, response: Response<User>) {
                if (response.isSuccessful) {
                    val user = response.body()
                    val intent = Intent(applicationContext, MainActivity::class.java)
                    intent.putExtra( name: "loggedInUserId", user?.id)
                    startActivity(intent)
                }
                else if(response.code() == 400){
                    toast( text: "User with given email does not have an account yet")
                }
                else if(response.code() == 401) {
                    toast( text: "Wrong password")
                }
            }

            override fun onFailure(call: Call<User>, t: Throwable) {
                t.printStackTrace()
                Log.e( tag: "error", t.message.toString())
            }

        })
    }
}

private fun redirectToRegisterActivity() {
    val intent = Intent( packageContext: this, RegisterActivity::class.java)
    startActivity(intent)
}
```

```
private fun register() {
    val editTextFirstName = findViewById<EditText>(R.id.editTextFirstNameR)
    val editTextLastName = findViewById<EditText>(R.id.editTextLastNameR)
    val editTextMail = findViewById<EditText>(R.id.editTextMailR)
    val editTextPassword = findViewById<EditText>(R.id.editTextPasswordR)
    val editTextConfirmPassword = findViewById<EditText>(R.id.editTextConfirmPassword)

    val firstName = editTextFirstName.text.toString()
    val lastName = editTextLastName.text.toString()
    val email = editTextMail.text.toString()
    val password = editTextPassword.text.toString()
    val confirmPassword = editTextConfirmPassword.text.toString()

    val user = User(email, password, firstName, lastName)
    val call = serviceGenerator.register(user)

    if(firstName.isEmpty() || lastName.isEmpty() || email.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()) {
        toast( text: "All the fields have to be filled")
    }
    else if(password != confirmPassword) {
        toast( text: "Passwords don't match")
    }
    else {
        call.enqueue(object : Callback<User> {
            override fun onResponse(call: Call<User>, response: Response<User>) {
                if (response.isSuccessful) {
                    toast( text: "Successfully created an account")
                    val intent = Intent(applicationContext, LoginActivity::class.java)
                    startActivity(intent)
                }
                else if(response.code() == 400){
                    toast( text: "User with given email already has an account")
                }
            }

            override fun onFailure(call: Call<User>, t: Throwable) {
                t.printStackTrace()
                Log.e( tag: "error", t.message.toString())
            }

        })
    }

}
```

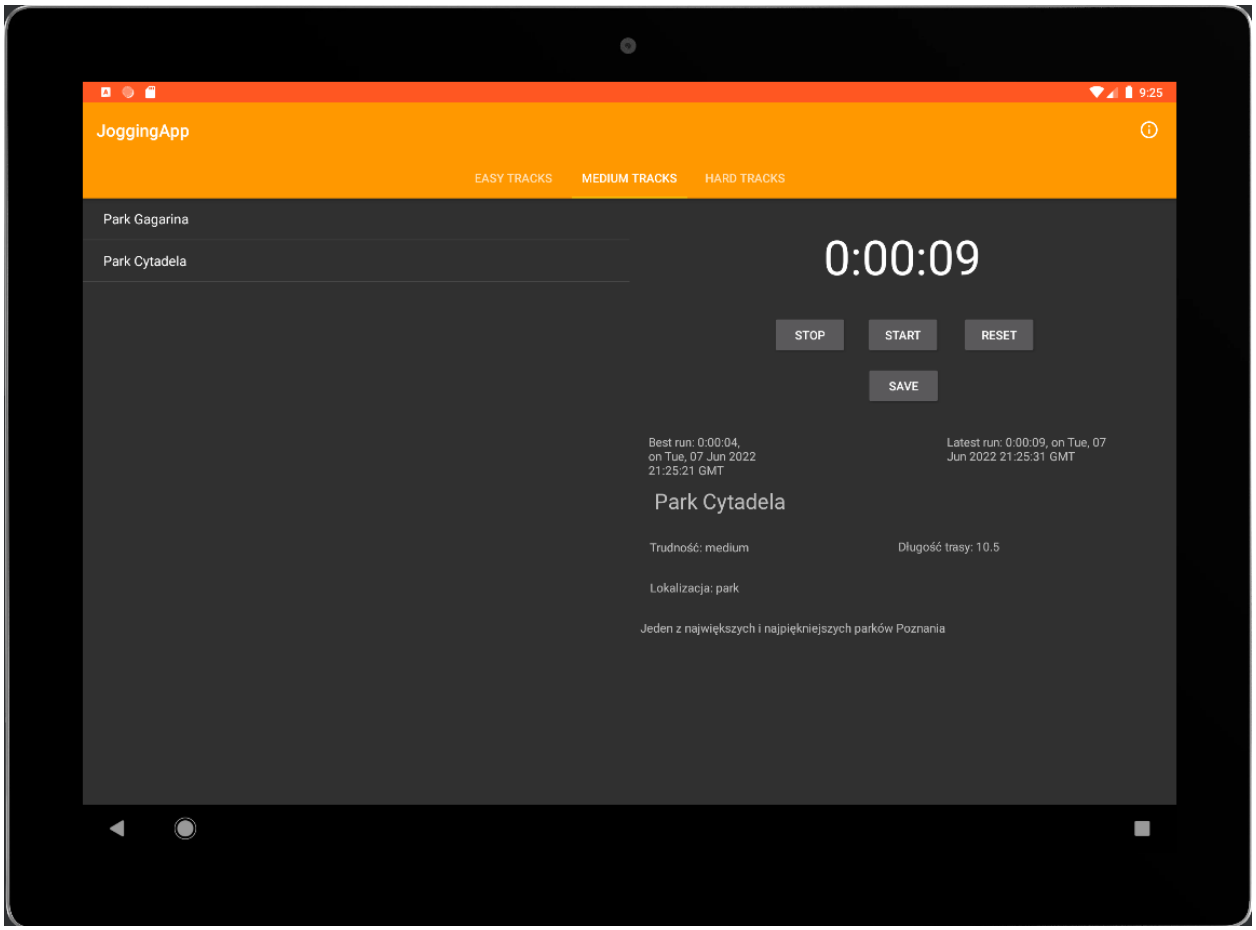
Listy tras

Informacje o trasach są pobierane poprzez zapytania http do api zwracającego odpowiednie dane. W strukturze bazy danych każdy rekord trasy zawiera informacje o jej długości, lokalizacji, trudności oraz krótki słowny opis. Informacje o poziomie trudności trasy są wykorzystane przez ViewPagera, który dzieli widok na trzy sekcje, pomiędzy którymi można poruszać się poprzez naciśnięcie odpowiednich nagłówków, a także poprzez gesty. Każda widok listy to osobny fragment wywoływany z odpowiednim parametrem.




```
private class SectionsPagerAdapter(fm: FragmentManager?, loggedInUserId: Int) : FragmentPagerAdapter(fm!!) {
    val loggedInUserId = loggedInUserId

    override fun getCount(): Int {
        return 3
    }

    override fun getItem(position: Int): Fragment {
        var trackListFragment = TrackListFragment()
        when (position) {
            0 -> {
                trackListFragment.trackDifficultyName = "easy"
            }
            1 -> {
                trackListFragment.trackDifficultyName = "medium"
            }
            2 -> {
                trackListFragment.trackDifficultyName = "hard"
            }
        }
        trackListFragment.loggedInUserId = this.loggedInUserId
        return trackListFragment
    }

    override fun getPageTitle(position: Int): CharSequence? {
        when (position) {
            0 -> return "Easy Tracks"
            1 -> return "Medium Tracks"
            2 -> return "Hard Tracks"
        }
        return null
    }
}
```

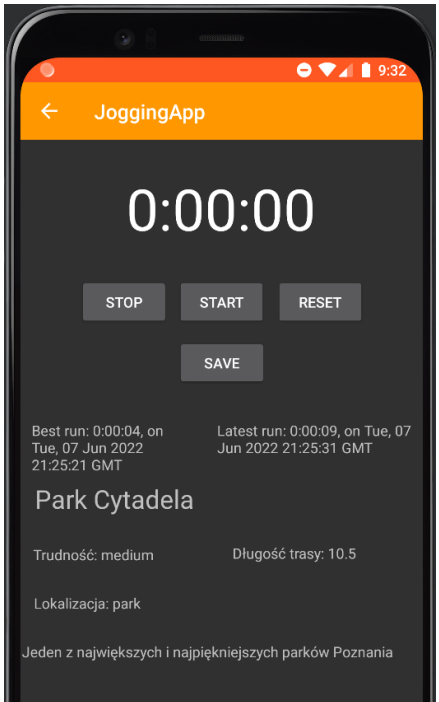
Naciśnięcie na poszczególne elementy listy przenosi do widoku szczegółów danej trasy. W wersji na smartfony użytkownik jest przenoszony do osobnej aktywności zawierającej fragment ze szczegółami, natomiast w wersji na tablety fragmenty listy i szczegółów są umieszczone w tej samej aktywności.



```kotlin
override fun itemClicked(id: Int) {

    var fragmentContainer = findViewById<FrameLayout>(R.id.fragmentContainer)
    if (fragmentContainer != null) {
        val trackDetailsFragment = TrackDetailsFragment()
        trackDetailsFragment.setTrack(id)
        trackDetailsFragment.setLoggedInUserId(loggedInUserId)

        var transaction = supportFragmentManager.beginTransaction()
        transaction.replace(R.id.fragmentContainer, trackDetailsFragment)
        transaction.addToBackStack( name: null)
        transaction.setTransition(TRANSIT_FRAGMENT_FADE)
        transaction.commit()
    }
    else {
        val intent = Intent( packageContext: this, TrackDetailsActivity::class.java)
        intent.putExtra(TrackDetailsActivity.TRACK_ID_KEY, id)
        intent.putExtra( name: "loggedInUserId", loggedInUserId)
        startActivity(intent)
    }
}
```

Szczegóły trasy

W widoku szczegółów użytkownik może zapoznać się z dokładnymi danymi o trasie i zmierzyć czas, w którym ją pokonał. Wyniki są zapisywane do bazy danych, dzięki czemu można w każdej chwili zobaczyć swój rekord i ostatni czas na danej trasie. Czas jest mierzony przez prosty stoper odmierzający czas z dokładnością do 1 sekundy, znajdujący się we fragmencie dynamicznym.



Podobnie jak w przypadku ekranu rejestracji, z widoku szczegółów trasy można powrócić do nadrzędnego widoku listy za pomocą strzałki w toolbarze.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    if (savedInstanceState == null) {
        val stopperFragment = StopperFragment()
        stopperFragment.loggedInUserId = loggedInUserId
        stopperFragment.trackId = trackId

        var transaction = childFragmentManager.beginTransaction()
        transaction.add(R.id.stopperContainer, stopperFragment)
        transaction.addToBackStack( name: null)
        transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
        transaction.commit()
    }
    else {
        trackId = savedInstanceState.getInt( key: "trackId")
        loggedInUserId = savedInstanceState.getInt( key: "loggedInUserId")
        this.setTrack(trackId)
        this.setLoggedInUserId(loggedInUserId)
    }
}
```

```
fun setTrack(trackId: Int) {
    this.trackId = trackId
    val serviceGenerator = ServiceGenerator().buildService(ApiService::class.java)
    val call = serviceGenerator.getTrackById(trackId)

    call.enqueue(object : Callback<Track> {
        @SuppressLint("SetTextI18n")
        override fun onResponse(
            call: Call<Track>,
            response: Response<Track>
        ) {
            if (response.isSuccessful) {
                track = response.body()!!

                if(view != null) {
                    val trackNameTV = view?.findViewById<TextView>(R.id.trackDetailsTrackName)
                    val trackLengthTV = view?.findViewById<TextView>(R.id.trackDetailsTrackLength)
                    val trackDifficultyTV = view?.findViewById<TextView>(R.id.trackDetailsTrackDifficulty)
                    val trackLocalizationTV = view?.findViewById<TextView>(R.id.trackDetailsTrackLocalization)
                    val trackDescriptionTV = view?.findViewById<TextView>(R.id.trackDetailsTrackDescription)

                    trackNameTV?.text = track.name
                    trackLengthTV?.text = "Długość trasy: " + track.length.toString()
                    trackDifficultyTV?.text = "Trudność: " + track.difficulty_name.toString()
                    trackLocalizationTV?.text = "Lokalizacja: " + track.localization_name.toString()
                    trackDescriptionTV?.text = track.description.toString()
                }
            }
        }

        override fun onFailure(call: Call<Track>, t: Throwable) {
            t.printStackTrace()
            Log.e( tag: "error", t.message.toString())
        }
    })
}
```

```
private fun getLatestRunTime() {
    val call = serviceGenerator.getLatestUserRun(trackId, loggedInUserId)

    call.enqueue(object : Callback<Run> {


        @SuppressLint("SetTextI18n")
        override fun onResponse(call: Call<Run>, response: Response<Run>) {
            if (response.isSuccessful) {
                val res = response.body()
                latestRunTime = res!!.time
                val hours: Int = latestRunTime / 3600
                val minutes: Int = latestRunTime % 3600 / 60
                val secs: Int = latestRunTime % 60
                val clockTime = String.format("%d:%02d:%02d", hours, minutes, secs)
                val latestRunDate = res!!.date

                val latestRunTextView = view?.findViewById<TextView>(R.id.latestRunTextView)
                latestRunTextView?.text = "Latest run: " + clockTime + ", on " + latestRunDate
            }
            else if (response.code() == 401) {
                val latestRunTextView = view?.findViewById<TextView>(R.id.latestRunTextView)
                latestRunTextView?.text = "------"
            }
        }

        override fun onFailure(call: Call<Run>, t: Throwable) {
            t.printStackTrace()
            Log.e( tag: "error", t.message.toString())
        }


    })
}
```