Katedra
Informatyki i Automatyki
Politechniki Rzeszowskiej

Sprawozdanie końcowe

Projekt Grzybek

Opracował: L05 156313

Spis treści

Praca w zespole 3

Za co odpowiadałem w projekcie 3

jaki był mój okres aktywności w projekcie 3

Co wykonałem w projekcie 3

Zespół, a praca 8

Jak przebiegała praca w zespole 8

za co w projekcie odpowiadali inni 8

za co w projekcie odpowiadali Project Manager'zy 8

za co w projekcie powinni odpowiadać Project Manager'zy 8

za co w projekcie odpowiadali interesariusze 9

za co w projekcie powinni odpowiadać interesariusze 9

Praca w zespole

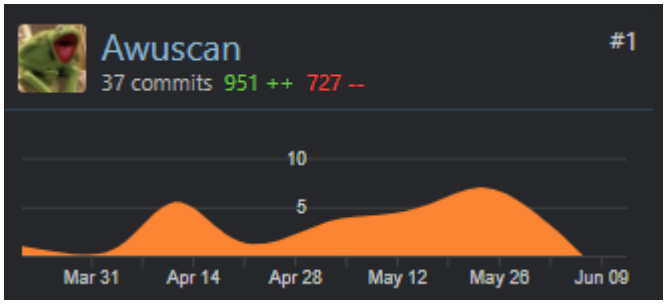
ZA CO ODPOWIADAŁEM W PROJEKCIE

Głównymi zadaniami, z którymi przyszło mi się zmierzyć to:

- Obsługa logowania po stronie aplikacji
- Stworzenie funkcji pobierających dane
- Odświeżanie danych poprzez gest pociągnięcia listy w dół
- Utworzenie modelu bazy danych
- Pomoc przy projektowaniu aplikacji
- Naprawianie bugów i przeróżne poprawki

JAKI BYŁ MÓJ OKRES AKTYWNOŚCI W PROJEKCIE

Od początku do końca.



CO WYKONAŁEM W PROJEKCIE

Obsługa logowania CAS

Uwierzytelnianie studenta poprzez CAS może odbywać się jedynie w domenie Politechnik Rzeszowskiej, tutaj z pomocą przyszedł nasz prowadzący Maciej Penar udostępniając nam proxy umożliwiające odebranie odpowiedzi systemu CAS po zalogowaniu.

Logowanie odbywa się poprzez utworzone WebView:

```
<WebView
android:id="@+id/web_view"
android:layout_width="match_parent"
android:layout_height="match_parent" android:visibility="invisible">
</WebView>
```

Odpowiedz CAS w przypadku podania poprawnych danych przesyła w odpowiedzi zakodowane w Base64 dane użytkownika. Dane te znjadują się adresie proxy na które zostało wykonane przekierwonie. W celu pobrania tych danych utworzono została klasa *URLInterceptor* dziedzicząca po *WebViewClient,* została ona podpieta po WebView.

```
web_view.apply {
    webViewClient = URLInterceptor()
}
```

URLInterceptor wykonuje różne zadania na podstwie adresu strony wyświetlanej w WebView.

```
override fun onPageStarted(view: WebView?, url: String?, favicon: Bitmap?) {
    super.onPageStarted(view, url, favicon)
    if (url!!.contains("www.mpenar.kia.prz.edu.pl/?response=")) {
        tryLoginUser(view, url)
    }
    if (!url!!.contains(CASurl)) {
        view?.visibility = View.GONE
        b?.startAnimation(AnimationUtils.loadAnimation(main,R.anim.rotate))
    }
}
```

Gdy zostanie wykryte opuszczenie strony logowania CAS widok WebView na czas weryfikacji logowania zostaje ukryty i uruchamia się animacja logowania. W przypadku wykrycia adresu proxy z odpowiedzą CAS uruchamiana jest funkcja *tryLoginUser.*

```
private fun tryLoginUser(view: WebView?, url: String) {
    if (checkAccess(url)){
        transition.start()
        main.login(getResponse(url))
    }else{
        view?.visibility = View.VISIBLE
        view!!.loadUrl(CASurl)
    }
}

private fun checkAccess(url: String) : Boolean{
    return !url.contains("Access%20Forbidden")}

private fun getResponse(url: String) : String {
    return url.substringAfter("response=")}
```

W przypadku, gdy nie uzyskano dostępu przywracany jest widok WebView oraz ponownie zostaje załadowanastrona logowania. W przeciwnym przypadku, gdy adres zawiera odpowiedz w formie ciągu znaków zakodowanych w Base64 uruchamiana zostaje główna funkcja logowania.

W tym miejscu napotkany został problem w starszych wersjach androida – nie działało drugie przekierowanie sytemu CAS – przekierowanie na stronę proxy. Widok WebView zatrzymywał się na stronie CAS zawierającej odpowiedz w ciele strony. W celu obejścia tego problemu stworzony został interfejs JS duplikujący funkcjonalność *URLInterceptor.*

```
internal class MyJavaScriptInterface {
@JavascriptInterface
    fun processHTML(html: String) {
    var document = Jsoup.parse(html)
    val divs = document.select("div")
        tryLoginUser(webview,divs[0].ownText())
    }

    private fun checkAccess(div: String) : Boolean{
        return !div.contains("Access%20Forbidden")
    }

    private fun getResponse(div: String) : String {
        return div.substringAfter("response=")
    }

    private fun tryLoginUser(view: WebView?, div: String) {
        if (checkAccess(div)){
            main.login(getResponse(div))
        }else{
            view!!.visibility = View.VISIBLE
            view!!.loadUrl(CASurl)
        }
}}
```

W klasie *URLInterceptor* dodana została funkjca wykrywajaca ten problem i uruchamiają fukcję logowania z interfesju JS.

```
override fun onPageFinished(view: WebView, url: String) {
    if(url!!.contains("www.mpenar.kia.prz.edu.pl/casproxy.php?redirect=http://
    www.mpenar.kia.prz.edu.pl&key=ed5fddea-9be9-4955-9718-fb429fed17f9&ticket=")){
        view.loadUrl("javascript:window.HTMLOUT.processHTML('<head>'+
        document.getElementsByTagName('html')[0].innerHTML+'</head>');")
        view.visibility = View.GONE
    }
}
```

Gdy już zostanie wykryta poprawna odpowiedz uruchamiana jest fukcja dekodowania odpowiedzi i logowania poprzez Firebase stworzona przez Marcina Pazowskiego.

Funkcje pobierania danych

Funkcje pobierania danych z znajdują się w 3 widokch aplikacji:

Ekran główny – Najbliższe wydzarzenia

```
private fun getEvents() {

    val today = Calendar.getInstance()
    today.add(Calendar.MINUTE,-120)
    var futureDate = Calendar.getInstance()
    futureDate.add(Calendar.DATE, 7)

    FirebaseFirestore.getInstance().collection("Events")
        .whereGreaterThanOrEqualTo("DateStart",Timestamp(today.time))
        .whereLessThanOrEqualTo("DateStart",Timestamp(futureDate.time))
        .get()
        .addOnSuccessListener { documents ->
            for (document in documents) {

                val date : Timestamp = document.data["DateStart"] as Timestamp
                val calendar = Calendar.getInstance()
                calendar.time = date.toDate()
                var nextEvent = DataForEvents(
                document.id,
                document.data["Name"] as String,
                document.data["Desc"] as String,
                calendar as Calendar,
                document.data["Owner"] as String
                )
                eventsList.add(nextEvent)
                events_recycleview.adapter = eventsAdapter
                eventsAdapter.notifyDataSetChanged()
            }
    }
    .addOnFailureListener { exception ->
    Log.w(TAG, "Error getting documents: ", exception)
    }

    swipeContainer.isRefreshing = false
}
```

Dane pobierane są przy uwzględnieniu zakresy daty i godziny rozpoczęcia. Pobierane zostają wydzarzenia od dwóch godzin wstecz i 7 dni do przodu od aktualnej daty i godziny. Dane o wydarzeniach zapisywane są w obiektach *DataForEvents*, które następnie dodawane są do listy *RecycleView* której adapeter zostaje powiadomiony o nowym elemencie.

Ekran – Moje wydarzenia

Tutaj funkjca wygląda bardzo podobnie, jedyną zmianą jest argument dla pobrania danych.

```
FirebaseFirestore.getInstance().collection("Events")
    .whereEqualTo("Owner", FirebaseAuth.getInstance().currentUser!!.uid)
    .orderBy("DateStart")
    .get()
```

Pobrane zostają wydarzenia utworzone przez aktualnie zalogowanego użytkownika, które zostaną posortowane według daty.

Aby można było posortować dane wg daty utworzony został w bazie Cloud Firestore indeks.

Collection ID	Fields indexed	Query scope	Status
Events	Owner Ascending DateStart Ascending	Collection	Enabled

Ekran – Kalendarz

Funkja pobrania danych także wygląda podobnie jak w poprzedich widokach.

Zmienione zostały argumenty. Pobrane zostają wydarzenia z całego aktualnie wyświtelaneggo w kalendarzu miesiąca. przed pobraniem danych kalendarz zostaje wyczyszczony.

```
fun getEvents(firstDayOfMonth: Calendar){

compactCalendarView!!.removeAllEvents()

var lastDayOfMonth = Calendar.getInstance()
    lastDayOfMonth.set(Calendar.YEAR,firstDayOfMonth.get(Calendar.YEAR))
    lastDayOfMonth.set(Calendar.MONTH,firstDayOfMonth.get(Calendar.MONTH)+1)
    lastDayOfMonth.set(Calendar.DAY_OF_MONTH,1)
    lastDayOfMonth.set(Calendar.HOUR,0)
    lastDayOfMonth.set(Calendar.MINUTE,0)
    lastDayOfMonth.set(Calendar.SECOND,0)
    lastDayOfMonth.set(Calendar.MILLISECOND,0)

db.collection("Events")
        .whereGreaterThanOrEqualTo("DateStart",Timestamp(firstDayOfMonth.time))
        .whereLessThanOrEqualTo("DateStart",Timestamp(lastDayOfMonth.time))
        .get()
```

Każde z pobranych wydarzeń zostaje dodane do kalendarza.

```
compactCalendarView!!.addEvent(Event(Color.rgb(255, 0, 0), event.date.timeInMillis,
event))
```

Odświeżanie danych poprzez gest pociągnięcia listy w dół

W celu obsługi gestu lista wydarzeń w *RecycleView* widoku *Najbliższe wydzarzenia* oraz *Moje wydarzenia* został opkaowany przez widget *SwipeRefreshLayout.*

```
<android.support.v4.widget.SwipeRefreshLayout
android:id="@+id/swipeContainer"
android:layout_width="match_parent"
android:layout_height="match_parent">
<android.support.v7.widget.RecyclerView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/myEventsRecyclerView"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"/>
</android.support.v4.widget.SwipeRefreshLayout>
```

W obu widokach dodano fukcję do obsługi gestu.

```
swipeContainer.setOnRefreshListener {
eventsAdapter.eventsList.clear()
    getEvents()
}
```

Po wykryciu gestu wywoływana zostaje ponownie funkcja pobrania danych *getEvents.*

Ustawiony został także kolor elementu ukazującego proces odświeżania.

```
swipeContainer.setColorSchemeResources(
    R.color.primaryColor
)
```

Zespół, a praca

JAK PRZEBIEGAŁA PRACA W ZESPOLE

Praca w zespole przebiegała sprawnie. Każdy z członków dostał swoje zadania do wykonania.

ZA CO W PROJEKCIE ODPOWIADALI INNI

Marcin Pazowski:

- obsługa dekodowania odpowiedzi CAS'a i logowania użytkownika po stronie Firebase'a

Artur Przystaś:

- komunikacja z PM'ami
- funkcja dodawania wydarzeń

Bartłomiej Nawój:

- komunikacja z PM'ami
- funkcja powiadomień push o nowych wydarzeniach
- integracja z Firebase
- integracja z Fabric

Maciek Miśkowiec:

- logo

Mateusz Nehrebecki:

- tworzenie widoków
- design aplikacji zgodny z Material Design

Rafał Piszko:

- stworzenie bazy projektu
- tworzenie widoków
- stworzenie rozwijanego menu

ZA CO W PROJEKCIE ODPOWIADALI PROJECT MANAGER'ZY

Stworzyli bardzo ogólną wizję produktu. Otrzymaliśmy jedynie wizję jak aplikacja ma działać oraz jej główne funkcje.

ZA CO W PROJEKCIE POWINNI ODPOWIADAĆ PROJECT MANAGER'ZY

Za stworzenie szczegółowej wizji gotowego produktu. Utworzenie makiet widoków aplikacji, szczegółowy opis funkcjonalności, jakie dane powinna przetrzymywać baza danych.

Powinni też w jakimś stopniu rozliczać zespół z wyników pracy. Wiąże się to także z argumentowaniem funkcjonalności aplikacji. Przy pytaniu do zespołu, dlaczego nadal nie ma jednej z funkcjonalności wystarczającym dla PM'ów argumentem z naszej strony było: „Bo za trudne".

Swoją obecnością PM'i nie wnieśli do projektu nic a tylko go opóźnili poprzez wielogodzinne spotkania, które nic nie zmieniły w projektu.

ZA CO W PROJEKCIE ODPOWIADALI INTERESARIUSZE

Prowadzący przedmiot po stronie wydziału informatyki służył pomocą, co do wyboru technologii użytych w procesie tworzenia aplikacji oraz udostępnił zespołowi możliwość logowania się przez system CAS. W późniejszym okresie organizował spotkanie, na których analizował kod oraz wygląd aplikacji a także poruszane były napotkane problemy.

Nie jest mi znane, za co odpowiadał prowadzący z wydziału zarządzania, widziałem go jedynie raz na prezentacjach projektów.

ZA CO W PROJEKCIE POWINNI ODPOWIADAĆ INTERESARIUSZE

Prowadzący z wydziału informatyki w odpowiednim stopniu uczestniczył w procesie tworzenia aplikacji.

Prowadzący z wydziału zarządzania powianiem uczestniczyć w spotkaniach PM'ów z szefami zespołów (przynajmniej raz) oraz rozliczać pracę i zaangażowanie swoich studentów.