Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики і обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота

з предмету

«Інтеграційні програмні системи»

Виконали: студенти 4 курсу
ФІОТ, ІО-41
Смішний Дмитро
Смішний Денис
Логвинчук Андрій

2017

1. Опис проекту

Загалом, даний проект є дещо спрощеним аналогом такого сервісу, як Badoo. А саме, кожен з зареєстрованих користувачів має свій профіль в системі, де вказані його контактні дані, його нікнейм, короткий опис самого користувача та фотографія аватару. Користувач може знаходити людей за нікнеймов в пошуку, подавати запити на дружбу, переглядати профілі інших користувачів а також спілкуватися з ними через вбудований в клієнт чат. Однак, основною фішкою все ж залишається відображення активних користувачів на карті. Це дозволяє побачитися зі своїм віртуальним другом наживо, наприклад.

Мобільний клієнт: Клієнт для ОС Android, що взаємодіє із віддаленим сервером через REST-запити, а той, свою чергу, відправляє нам інформацію у вигляді JSON. Зв’язок із сервером здійснюється через бібліотеку Retrofit 2.0, а парсинг отриманого JSON в JavaBean відбувається через бібліотеку GSON. Зображення, відображаються за допомогою Picasso. Для керування залежностями в проекті використовується Dagger. Для кращої структуризації і модуляції додатку було використано бібліотеку RxJava.

Для кращої оптимізації роботи, додаток побудований на фрагментах, що дозволяє економити ресурси процесора і оперативної пам'яті моб. пристрою.

Щодо дизайну, то в процесі розробки було прийняте рішення не відходити від концепції Material Design від Google. Тому було використано стандартний палет кольорів для додатка, а основним шаблоном дизайну є Navigation Drawer.

Серверна частина: сервер обробляє API-запити клієнта. Сервер має класичну трирівневу артхутектуру: контролер/бізнес-логіка/доступ до данних. Для обробки HTTP-запитів використовується фреймворк Spring MVC, маршалінг об’єктів предметної області у JSON-представлення виконується бібліотекою Jackson, а доступ до бази даних - ORM-системою на основі Java Persistence API. HTTP Basic аутентифікація мала здійснюється засобами Spring Security.

База даних запускається в окремому Docker-контейнері, дані знаходяться на окремому томі, що монтується з локальної файлової системи. Це дає змогу швидко розгортати сервер на чистому середовищі, оскільки усі ролі і доступи вже налаштовані в контейнері, а дані можна швидко мігрувати з резервної копії.

Також були написані Gradle-завдання для зручності збірки та запуску Docker контейнера з сервером.

Примітка: проект реалізований ЧАСТКОВО.

2. Система автоматичної збірки. Gradle

Gradle – це система автоматичної збірки, що побудована на принципах Apache Ant і Apache Maven. Втім, для конфігурації проекту тут використовують предметно-орієнтовану мову (DSL) Groovy замість традиційної XML. На відміну від Apache Maven, заснованого на концепції життєвого циклу проекту, і Apache Ant, в якому порядок виконання завдань (targets) визначається відношеннями між залежностями (depends-on), Gradle використовує спрямований ациклічний граф для визначення порядку виконання завдань.

Gradle був розроблений для розширюваних, багато проектних збірок, і підтримує інкрементальні збірки. Він визначає, які компоненти дерева збірки не змінилися і які завдання, що залежать від цих частин, не вимагають перезапуску.

Основні плагіни призначені для розробки і розгортання Java, Groovy і Scala додатків, але готуються плагіни і для інших мов програмування.

Причинами використання саме Gradle під час розробки проекту стало те, що саме ця система збірки використовується як стандарт для мобільних додатків під ОС Android. Більше того, Groovy є зручною мовою для опису конфігураційних файлів системи збірки, а сам Gradle дозволяє збирати як власне увесь проект загалом, так і його компоненти окремо (мова йде про серверну частину і мобільний додаток, які в сукупності є цілісним проектом).

3. Сервер безперервної інтеграції. Travis-CI

Як виявилося, термін «continuous integration» досить старий. Він був введений Мартіном Фаулером (Martin Fowler) у 2000-му році і викладений у статті «Continuous Integration» і українською звучить як «безперервна інтеграція». Це частина процесу розробки, в якій розробляється проект збирається/тестується в різних середовищах виконання автоматично і безперервно. Задумувалася дана методика для найшвидшого виявлення помилок/протиріч інтеграції проекту, а отже зниження витрат на наступні простої.

Принцип досить простий: на окремій машині працює якась служба, в обов'язки якої входить отримання вихідного коду проекту, його збірка, тестування, логування, а також можливість надати для аналізу дані виконання перерахованих операцій.

Втім, такий підхід має і певні недоліки. А саме, потрібно: виділити окремий сервер і підтримувати його в робочому стані, забезпечити наявність необхідних програмних комплексів, налаштувати середовища виконання, робити резервні копії даних і т.д. Все це вимагає чимало часу і ресурсів. І цілком логічним є можливість делегувати цю відповідальність на сторонні сервіси. От якраз таким і є Travis-CI - «хостинг безперервної інтеграції для open source співтовариства».

Короткий перелік завдань, які виконує Travis-CI:

- Проблеми продуктивності, пов’язані з розміткою інтерфейсу
- Невикористані ресурси
- Невідповідності розмірів масивів (коли масиви визначені у множинних конфігураціях).
- Проблеми доступності та інтернаціоналізації («магічні» рядки, відсутність атрибуту contentDescription і т.д)
- Проблеми з іконками (невідповідності розмірів, порушення DRY)
- Проблеми зручності використання (Наприклад, не зазначений спосіб введення для текстового поля)
- Помилки в маніфесті

4. Експоненціальна витримка (Exponential Backoff)

Для вирішення задачі раптового зникнення з'єднання мобільного клієнта з бек-ендом було використано підхід експоненціальної витримки, що передбачає повторні запити з тими ж параметрами, але через неоднакові проміжки часу. Наприклад, якщо в ході першого запиту користувач отримує помилку, то додаток намагається знову звернутися до сервера, але уже через більший проміжок часу. Так доти, доки не отримаємо позитивну відповідь, або не вичерпаємо дозволену кількість ітерацій на повторні запити (в даному випадку загальна кількість запитів - 5), збільшуючи з кожним разом інтервал між ними.

Даний підхід чудово вписується в технологію RxJava. Для реалізції був використаний клас ExponentialBackoff (https://github.com/iodudes/friends-nearby/blob/%239-Creating_basic-UI-controllers/friendly/app/src/main/java/com/denshiksmle/friendly/util/ExponentialBackoff.java, credits: http://kevinmarlow.me/better-networking-with-rxjava-and-retrofit-on-android/) з відповідним методом, та, за домопогою методу retryWhen, інтегрований в запити користувача.

```
@Inject
public RegistrationScreenPresenter(@NonNull final Retrofit retrofit,
                                   @NonNull final RegistrationScreenContract.RegistrationView mRegistrationView) {
    this.retrofit = retrofit;
    this.mRegistrationView = mRegistrationView;
}

@Override
public void registerUser(@NonNull String email, @NonNull String password, @NonNull String userName) {
    retrofit.create(UserService.class).getUser(email, password)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .unsubscribeOn(Schedulers.io())
            .retryWhen(ExponentialBackoff.exponentialBackoffForExceptions( initialDelay: 2,  numRetries: 4, TimeUnit.SECONDS, Exception.clas
            .subscribe(user -> mRegistrationView.registrationSuccess(user),
                    error -> mRegistrationView.registrationError(error.getMessage()));
}
```

В результаті,отриманий наступний графік