Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование
информационных систем

Липаев Савелий

Разработка сервиса для заказа еды

Учебная практика

Научный руководитель:
к. т. н., Литвинов Ю. В.

Санкт-Петербург
2020

Оглавление

1. Введение

Обучаясь в любом вузе страны (или школе) студенты (ученики) должны иметь возможность правильно питаться. В большинстве учебных заведений существуют столовые, которые находятся в тех же зданиях (или же рядом). Когда ученики приходят поесть в перерыве между занятиями, возникает огромная очередь у кассы (хотя во время занятий в столовой почти не бывает посетителей).

В результате чего многие студенты перестают посещать такие столовые, из-за нежелания тратить время в очереди, а потом есть в спешке — т.е. столовая теряет прибыль, а студент либо остаётся голодным, либо с большой вероятностью опаздывает на следующую пару.

Большинство современных кафе предоставляют различные приложения для предварительного заказа еды – это позволят сократить очереди и увеличить количество посетителей. Подобное приложение, адаптированное для студенческих столовых, было бы очень востребовано.

2. Постановка задачи

В данной семестровой были поставлены следующие задачи:

- Разработка мобильного приложения для предварительного заказа еды, на определённое время, в конкретной столовой, но без привязки к конкретному вузу.

- Написание RESTful API, обеспечивающий доступ к данным через запросы по сети и позволяющий максимально разделить фронтенд и бэкэнд, а так же увеличить производительность даже при относительно высокой нагрузке, благодаря следованию архитектурному стилю REST[13].

- Проектирование и реализация БД.

- Следование современным стандартам разработки ПО (виртуплизация с помощью Docker, swagger для документации)

- Применение практик DevOps и CI для поддержания проекта в рабочем состоянии на протяжении всего цикла разработки (Appveyor для Xamarin.Forms, Travis для ASP.NET, GitHub Actions)

- Написание веб приложения для кассиров.

Рис. 1: UI Сытый офис

3. Обзор

3.1. Обзор существующих решений

Самыми близкими аналогами являются рестораны, у которых есть возможность

3.2. Столовая Поварёшка

\+ Интуитивный, неперегруженный пользовательский интерфейс

— Узко специализированный, только для одной сети

3.3. Сытый офис

\+ Есть и веб-интерфейс и приложение для мобильных устройств

— Сервис работает только в двух городах

— Не адаптирован для студентов

3.4. KFC

\+ Простой и красивый веб-интерфейс

\+ Удобное мобильное приложение для iOS, Android

— Сервис ориентирован только на сеть рестаранов KFC

Рис. 2: Xamarin.Forms App Architecture

4. Реализация

4.1. Архитектура

В качестве основного фреймворка для написания мобильного приложения использовался Xamarin.Forms (далее XF). XF является отличным выбором для мультиплатформенной разработки.

Приложение, написанное на XF имеет единую логику для всех платформ написанную на языке C#, что тоже сыграло роль в выборе этого фреймворка, также он позволяет писать единый UI для всех платформ, но не запрещает нативной реализации всего UI или какой-то небольшой части.

Поскольку всё компилируется в нативный код мобильных платформ, приложения высокую XF имеют высокую производительность, очень близкую к нативной, а иногда и выше.[1]. В качестве архитектурного паттерна использовался Model-View-ViewModel(MVVM)[6], позволяющий разделить бизнес-логику и пользовательский интерфейс. MVVM является самым распространенным архитектурным паттерном для XF.

Для авторизации в приложение использовался открытый протокол авторизации OAuth 2.0, позволяющий мобильному приложению получать ограниченный доступ к аккаунту пользователя на различных сервисах. В приложении возможно проходить аутентификацию через Google и ВКонтакте.

Работает это следующим образом:

1. Приложение переадрессовывает на страницу авторизации сервиса
2. Пользователь авторизуется и подтверждает выдачу прав
3. Сервис переадрессовывает обратно в приложение, передавая приложению authorization code

[1]Performance Comparison: Xamarin.Forms, Xamarin.iOS, Xamarin.Android vs Android and iOS [8]

4. Приложение отправляет POST запрос с authorization code, в результате которого получает access token

5. С помощью токена приложение может получить данные о пользователе, которые он сам разрешил при подтверждении выдачи прав

Для работы с RESTful API использовали Refit, позволяющий легко описывать спецификацию в виде интерфейса с понятными выходными и входными параметрами, а также с возможностью манипулирования HTTP-заголовками для отдельных запросов.

При разработке мобильных приложений нужно учитывать нестабильное соединение с сетью, соответственно, возникает необходимость повторного выполнения запросов, не перегружая пользователя дополнительными просьбами повторить действие. Для этого используется библиотека **Polly**[9], с помощью которой удобно обрабатывать неудачные сетевые запросы.

4.2. ASP.NET Backend

4.2.1. Jwt and Identity

JWT(JSON web tokens) - это стандарт для кодирования данных, который включает в себя информацию о роли пользователя и другие данные, относящиеся к вашему сайту. - JWT предлагает безопасный метод отправки этой информации с использованием общего секрета. Разные маршруты могут требовать разных ролей, с перенаправлениями на другие страницы при неудачной аутентификации. Это означает, что вы можете использовать закодированный. Роль пользователя для управления содержимым и контроля пользовательских возможностей.

4.2.2. RESTfull API

Для кэширования информации о пользователе использовалось хранилище Akavache[1], работающее поверх SQLite3.

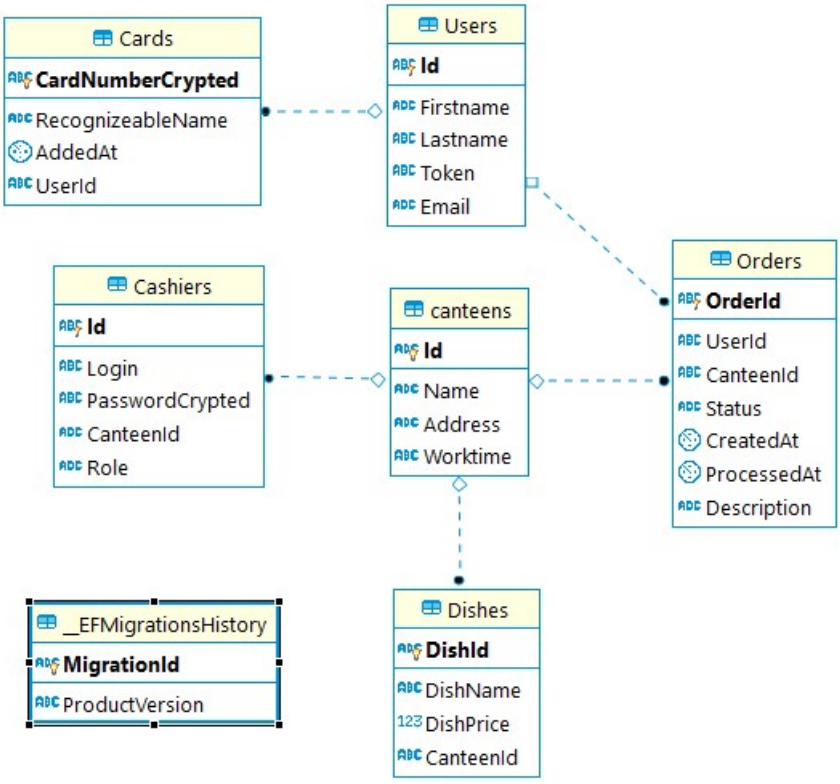

Рис. 3: База данных

Стоит отметить, что для кэширования можно было использовать другие мобильные СУБД, например Realm, но они были не так просты в использовании как Akavache и обладали избыточной функциональностью.

Для работы с СУБД PostgreSQL на бэкенде используется Entity Framework Core, схема базы данных представлена на рисунке 3.

Для оптимизации запросов к RESTfull API используется кэширование с помощью mem-cache бд Redis[5].

3.6) EF PostgreSQL, Redis PostgreSQL 3.7) Docker and CI(appveyor, travis) Для автоматизации тестиоования использовалмя AppVeyor и TravisCI что позволяло сразу же во время разработки получить готовую сборку, проверенную с помощью юнит-тестов 3.8) Vue.js – не готово!!!

3.9) Swagger Проблема в том, что REST не является само описательным протоколом. Это значит, что клиент должен знать конкретную комбинацию URL, HTTP метода и формата ответа. В некоторых случаях необходимо также знать также формат тела запроса. В любом слу-

чае, REST endpoints всегда должны быть описаны в одном конкретном документе, доступном для всех остальных разработчиков. Но Swagger — это не просто спецификация. Основная его мощь заключается в дополнительных инструментах.

5. Результаты

1. Получен опыт работы с фреймворком для создания мобильных приложени Xamarin.Forms
2. Разработано Мобильное приложение для заказа еды
3. Написан RESTfull API для получения данных
4. Реализованна схева авторизации пользователей с помощью протокола OAuth 2.0
5. Опыт работы с IdentityServer для ASP.NET
6. Получен опыт прототипирования интерфейса мобильного приложения
7. Получен опыт использования СУБД PostgreSQL, в связке с Redis, ORM EF
8. Опыт настройки веб сервера + devops(docker,ci,deploy)

Список литературы

[1] Akavache - Github репозиторий. — Access mode: https://github.com/reactiveui/Akavache.

[2] Chapsas Nick. ASP.NET Core 3/2.2 REST API Tutorial - Github репозиторий. — 2019. — Access mode: https://github.com/Elfocrash/Youtube.AspNetCoreTutorial.

[3] Cheung Harry. Mobile App Performance: J2ObjC vs Xamarin vs RoboVM vs RubyMotion. — 2015. — Access mode: https://medium.com/@harrycheung/cross-platform-mobile-performance-testing-d0454f5cd4e9.

[4] Comparing performance of Android apps written in Xamarin C# and Java - stackoverflow discussion. — 2013. — Access mode: https://stackoverflow.com/questions/17134522/does-anyone-have-benchmarks-code-results-comparing-performance-

[5] Distributed Redis Cache - Microsoft .NET Tutorial. — 2020. — Access mode: https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed?view=aspnetcore-3.1.

[6] MVVM паттерн. — 2019. — Режим доступа: https://ru.wikipedia.org/wiki/Model-View-ViewModel.

[7] NpSQL for Entity Framework Core - documentation. — Access mode: https://www.npgsql.org/efcore/.

[8] Performance Comparison: Xamarin.Forms, Xamarin.iOS, Xamarin.Android vs Android and iOS Native Applications. — 2017. — Access mode: https://www.altexsoft.com/blog/engineering/performance-comparison-xamarin-forms-xamarin-ios-xamarin-androi

[9] Polly - Github репозиторий. — Access mode: https://github.com/App-vNext/Polly.

[10] Refit - Github репозиторий. — 2020. — Access mode: https://reactiveui.github.io/refit.

[11] Битман Дмитрий. OAuth 2.0 простым и понятным языком. — 2011. — Режим доступа: https://habr.com/ru/company/mailru/blog/115163/.

[12] Скелет приложения для Xamarin - Github репозиторий. — 2019. — Access mode: https://github.com/Binwell/Order-King-Mobile-Core.

[13] Черников Вячеслав. Удобный REST для Xamarin-приложений. — 2016. — Режим доступа: https://habr.com/ru/company/microsoft/blog/310704/.