Контейнеры – хорошая альтернатива аппаратной виртуализации. Они позволяют запускать приложения в изолированном окружении, но при этом потребляют намного меньше ресурсов.

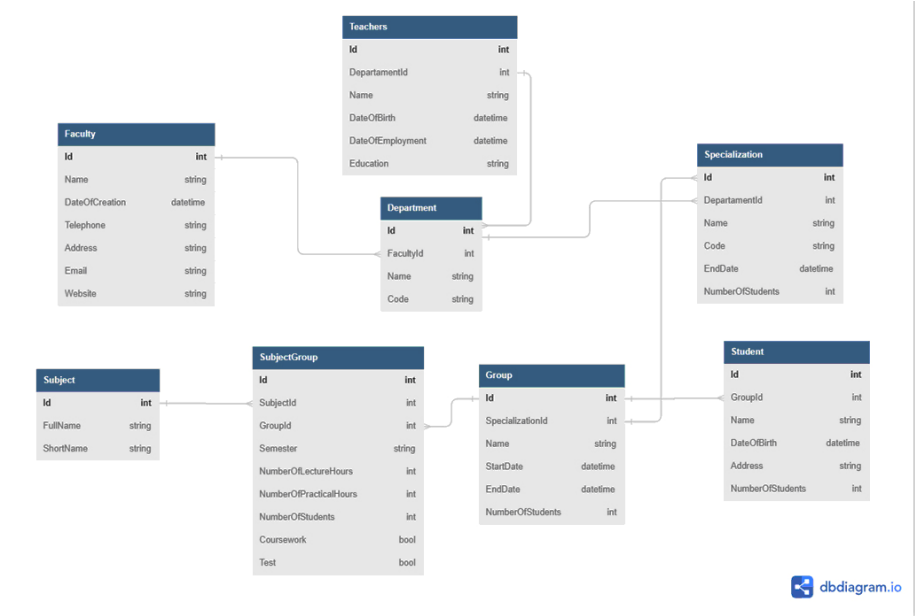

Рис. 2. Схема базы данных для управления информационным обеспечением задач кафедры

Таким образом, был разработан web-сервис, который получает данные из 1С базы данных университета и предоставляет данные в json. Данное приложение может быть использовано в учреждениях образования для автоматизации процессов обучения. Приложение позволит существенно упростить учебный процесс.

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ АВТОМАТИЗАЦИИ ВЕДЕНИЯ ИНДИВИДУАЛЬНОГО ПЛАНА ПРЕПОДОВАТЕЛЕЙ

Р. О. Езепенко

Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», Республика Беларусь

Научный руководитель Т. Л. Романькова

Описаны особенности разработки программного комплекса для автоматизации ведения индивидуального плана преподавателей с разновидным пользовательским интерфейсом; выявление функциональных и нефункциональных требований к приложению; обнаружение и дальнейшее объяснение особенности разработки компонентов пользовательского интерфейса, применяя Back-end for front-end подход.

Ключевые слова: автоматизация, микросервисы, *BFF*, *.NET*, *Android application*.

Процессы автоматизации в современном мире находят применение в различных сферах жизни исфера образования – не является исключением. В рамках данной работы приведен пример автоматизации ведения индивидуального плана преподавателя, с решением типовых задач, характерных для справочных систем.

Перед принятием решения, касаемых проектировки системы, таких, как выбор архитектуры, подход к разработке проекта, применяемые технологии, необходимо произвести детальный анализ предметной области и определить как функциональные, так и нефункциональные требования к проекту, оценить нагрузки, которые система должна выдерживать.

Список функциональных требований к приложению:

– предоставление и генерация шаблонных решений для индивидуального плана;

– автоматическая генерация нагрузки на полугодие;

– интеграция генерируемого отчета с *MSExcel*;

– валидация предоставляемого отчета;

– система уведомления пользователя;

– предоставление статистики по индивидуальным планам заведующему кафедры.

Список нефункциональных требований к приложению:

– применение контейнеризации при разработке компонентов приложения;

– интеграция оркестровки контейнеров (автоматическое размещение, координация и управление контейнерами);

– локализация приложения;

– разработка *WEB UI* и *Android Application*;

– разработка системы авторизации и аутентификации;

– проведение нагрузочного тестирования (проверка соответствия ожидаемых нагрузок на систему).

Было принято решение о разработке нескольких видов пользовательского интерфейса: *WEB UI* и *Android Application.* Данное решение связано с тем, что функционал, предоставляемый системой, будет иметь отличия для каждого пользовательского интерфейса. В связи с этим было решено отделить разработку серверной части и части пользовательского интерфейса. Разработка серверной части произведена в стиле *REST API*. Данный подход позволяет разделить клиентскую часть от серверной, а также подразумевает взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей – микросервисов, данное преимущество облегчает сопровождение и наращивание нового функционала приложения. В качестве технологии, с помощью которой разработана серверная часть, выбрана *ASP.NET Core*, к преимуществам которой можно отнести качественную документацию, кроссплатформенность, открытый исходный код, большое сообщество и поддержку в лице корпорации *Microsoft*.

При наличии нескольких видов пользовательского интерфейса возникает проблема: у нас есть функциональность на стороне сервера, которую мы хотим предоставить как через веб-интерфейс, так и через один или несколько мобильных интерфейсов. С системой, которая изначально была разработана с учетом веб-интерфейса, мы можем столкнуться с проблемой адаптации этих новых типов пользовательского интерфейса, из-за того, что у была тесная связь между веб-интерфейсом и существующими сервисами. Первым шагом в адаптации более чем одного типа *UI* обычно является предоставление единого серверного *API* и добавление дополнительных функций по мере необходимости для поддержки новых типов мобильного взаимодействия. Если эти разные пользовательские интерфейсы хотят выполнять одинаковые или очень похожие виды вызовов, то для такого типа универсальный *API* может легко добиться успеха. Однако природа мобильного опыта часто резко отличается от веб-интерфейса. Во-первых, возможности мобильного устройства очень разные. У нас меньше места на экране, а значит, мы можем отображать меньше данных. А во-вторых, характер взаимодействий, которые мы хотим обеспечить на мобильном

устройстве, может кардинально отличаться. Еще одна проблема с *API* общего назначения заключается в том, что они по определению предоставляют функциональные возможности нескольким приложениям, ориентированным на пользователя. Это означает, что единая серверная часть *API* может стать узким местом при добавлении нового функционала. Склонность *API* общего назначения брать на себя несколько обязанностей и, следовательно, требовать много работы, часто приводит к тому, что команда создается специально для обработки этой кодовой базы. Одно из решений этой проблемы, заключается в том, что вместо того, чтобы иметь *API* общего назначения, у нас есть один *API* для каждого вида пользовательского интерфейса. Концептуально мы должны рассматривать пользовательское приложение как два компонента: клиентское приложение, находящееся за пределами нашего периметра, и серверный компонент (*BFF*) внутри нашего периметра. На рис. 1 приведена схема *BFF* паттерна.

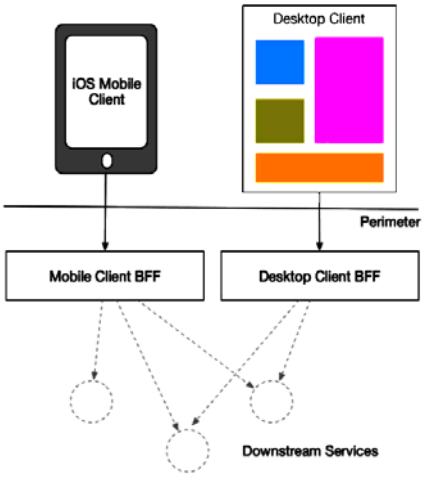

Рис. 1. Схема *BFF* паттерна

BFF тесно связан с конкретным пользовательским интерфейсом и обычно поддерживается той же командой, что и пользовательский интерфейс, что упрощает определение и адаптацию *API* в соответствии с требованиями пользовательского интерфейса, а также упрощает процесс согласования выпуска обоих компонентов. Исходя из этой особенности была разработана архитектура приложения, которая учитывает данную проблему. На рис. 2 приведена архитектура приложения.

Для реализации данной задачи выбрана авторизация и аутентификация на основе *JWT* токена, так как он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок с общей информацией по токену, полезные данные, такие как *id* пользователя, его роль и подписи.

Так как данные представляют из себя документ, который по мере заполнения обогащается данными, было принято решение о применении нереляционной базы данных *MongoDb*, к преимуществам которой можно отнести простую масштабируемость, гибкий *JSON*-формат документов, высокую производительность и быстрый доступ к данным.

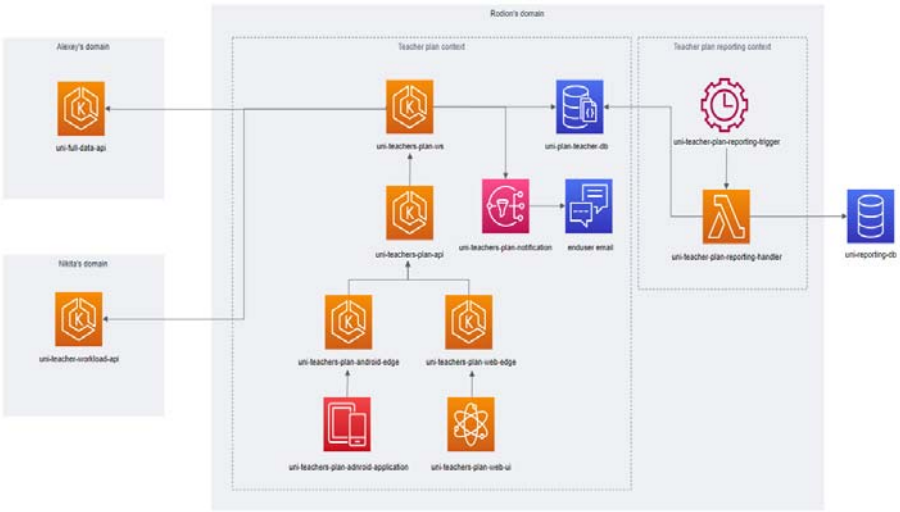

Рис. 2. Архитектура приложения

В заключение можно отметить тот факт, что разработанное приложение полностью выполняет поставленные как функциональные, так и нефункциональные требования. Обнаружена потенциальная проблема разновидного пользовательского интерфейса, с различием в предоставляемом функционале, решением которой является архитектура системы, учитывающая потенциальные проблемы системы.

WEB-ПРИЛОЖЕНИЕ ДЛЯ АВТОМАТИЗАЦИИ РАСПРЕДЕЛЕНИЯ НАГРУЗКИ ПО ПРЕПОДАВАТЕЛЯМ

Н. С. Заяц

Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», Республика Беларусь

Научный руководитель Т. Л. Романькова

Описаны компоненты, необходимые для создания программного комплекса по автоматизации распределения нагрузки: архитектура программного комплекса, схема базы данных, технологии, а также способы организации их взаимодействия.

Ключевые слова: автоматизация, документная база данных, контейнеризация, C#, Web API, REST, React.

В наши дни в независимости от занимаемой должности и сферы деятельности люди все чаще вынуждены документировать свою деятельность, составлять подробные планы. Это вызывается все большим усложнением рабочего процесса. Однако человеческие возможности ограничены, и дополнительная бумажная работа приводит к переработкам и стрессу.

Данная проблема встречается и в системе образования. Преподаватели вынуждены создавать, потом вручную перепроверять и, в случае наличия ошибки, переделывать большое количество документов. Конечно, подобные документы очень важны для организации работы в столь сложной организации, как университет. Из этого возникает объективная потребность в автоматизированной системе помощи преподавателям в составлении документации. Одним из таких документов является учебная нагрузка преподавателя.