Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Порсев Денис Витальевич

Сервис обучения персонала

Отчёт по учебной практике

Научный руководитель:
ст. преп. С.Ю. Сартасов

Санкт-Петербург
2021

Содержание

Введение

Цифровизация бизнеса в последние годы стала относиться к основным направлениям стратегического развития современных компаний. Ради сохранения лидерства в отрасли посредством увеличения эффективности деятельности предприятия и его работников большинство компаний обращается к разработчикам программного и аппаратного обеспечения с целью создания продуктов, которые смогут автоматизировать бизнес-процессы и помочь своим сотрудникам в выполнении привычных задач.

Одной из таких задач может стать планирование регулярных мероприятий по обучению персонала и повышению его компетенций.

Процесс создания и формирования расписания обучения персонала, особенно в крупных предприятиях - это трудоёмкая задача, требующая внимательного и строгого отношения к работе с большими данными. При этом сотрудники любой компании нуждаются в постоянном улучшении своих навыков и знаний, поэтому создание сервиса, который помог бы консолидировать информацию о персонале и сопроводить руководителя, занимающегося созданием этого графика, актуальная проблема для бизнеса.

Данная работа является частью проекта «Цифровой двойник энергетики», который компания ПАО «Газпром Нефть» планирует реализовать с целью интеграции новых технологий в работающие предприятия. Сервис обучения персонала разрабатывается двумя студентами СПбГУ как прототип одного из модулей этой системы, и возможность поучаствовать в этой разработке является для авторов отличным способом влиться в промышленное программирование и изучить новые технологии.

1. Постановка задачи

Целью работы является реализация и развертывание прототипа веб-приложения, главной задачей которого является помощь в отслеживании развития персонала предприятий компании Газпром Нефть.

Для достижения общей цели перед автором работы были поставлены следующие задачи:

1. Провести поиск аналогичных продуктов и выделить основные функциональные особенности.
2. Спроектировать и реализовать микросервис консолидации данных о персонале.
3. Разработать и реализовать дизайн интерфейса клиентской части сервиса, отвечающего за консолидацию данных о персонале.
4. Развернуть веб-приложение на виртуальном хостинге и провести апробацию среди конечных пользователей.

Задачи, поставленные перед другими участниками команды:

1. Проведение поиска аналогичных систем с целью уточнения функциональных требований.
2. Проектирование и реализация серверной части микросервиса обучающих событий.
3. Реализация клиентской части сервиса обучающих событий.
4. Реализация микросервиса авторизации и системы ролей пользователей приложения, а также клиентской части этого сервиса.
5. Проведение апробации приложения среди конечных пользователей.

2. Обзор существующих решений

В данной главе приведен обзор популярных продуктов, предназначенных для ведения кадрового учета и отслеживания навыков персонала. В ходе этого исследования также были уточнены функциональные требования к реализуемому продукту.

2.1 Обзор аналогов

Обзор аналогичных систем предусматривал исследование существующих программных продуктов и нахождение уже готовых решений поставленных перед нами задач.

2.1.1 1С: Предприятие

«1С: Предприятие» - это продукт компании «1С», который предлагает автоматизировать деятельность предприятий. Конкретный его компонент «1С:Зарплата и управление персоналом 8» позволяет работать с заявками на обучение, планом, а также сохраняет результаты обучения [1]. Список сотрудников в виде таблицы занимает основное окно программы, выше этого окна расположена панель разделов и инструментов.

При этом основным предназначением этого сервиса является расчёт зарплаты сотрудников, а не отслеживание прогресса в обучении и освоении новых навыков.

2.1.2 Microsoft Dynamics 365 Human Resources

Microsoft Dynamics 365 Human Resources - одно из бизнес-приложений, входящих в семейство продуктов Microsoft Dynamics 365, позволяющих планировать ресурсы предприятия. С помощью этого приложения можно настроить курсы, инструкторов, выставить цели и задачи перед сотрудниками [2]. В интерфейсе приложения каждому сотруднику присвоена карточка с подробной информацией. Руководители могут получить аналитику по кадрам в виде интерактивных графиков.

Однако семейство продуктов Microsoft Dynamics ориентировано в первую очередь на небольшие компании, чья административная структура обладает простым устройством. К тому же, с проектом Microsoft у российского пользователя могут возникнуть проблемы с локализацией, так как несмотря на то, что большая часть интерфейса переведена, некоторые функции (например, сведения о планировании) на данный момент доступны только на английском языке.

2.2 Сервис обучения

После рассмотрения аналогов были выявлены следующие особенности, которые в том или ином виде присущи такого рода приложениям: наличие страницы или карточки сотрудника с подробной информацией о нём, возможность выводить информацию о сотрудниках в виде списка или таблицы, возможность получения аналитики по кадрам в виде интерактивных графиков. Также системы имеют возможность планировать мероприятия по обучению, добавлять к ним персонал и отображать информацию о пройденных и назначенных курсах.

3. Обзор используемых технологий

В данном разделе описаны технологические инструменты, использованные при создании веб-приложения.

3.1 React

Для разработки интерфейса веб-приложения программисты в последние годы выбирают между двумя самыми популярными инструментами: JavaScript-библиотекой React и платформой для веб-разработки Angular.

Несмотря на то, что разработанный в Google Angular предоставляет большое количество возможностей в базовой установке, что упрощает изучение технологии и ускоряет процесс написания кода, эта особенность в то же время накладывает определенные ограничения на структуру проекта. Так, например, в микросервисной архитектуре некоторые предустановленные функции Angular могут оказаться лишними, и невозможность их удаления увеличит размер микросервиса. В то время как легковесная библиотека React, разработанная в Facebook, может быть настроена под нужды конкретного проекта и будет содержать используемую функциональность и не более того. К тому же, наличие JSX - HTML-подобного синтаксиса в React, компилируемого в JavaScript, - ускоряет процесс разработки и упрощает отладку приложения, указывая на ошибки во время компиляции. По этим причинам React был предпочтён в пользу Angular.

3.2 ASP.NET

Основной платформой, на которой разрабатывается веб-приложение, является набор инструментов ASP.NET, поддерживаемый Microsoft. А именно, используется одна из платформ, под названием ASP.NET MVC, предоставляющая набор шаблонов для реализации веб-приложений на основе паттерна MVC, который используется в данной разработке.

Более того, хорошая документация и поддерживаемые сообществом расширения библиотеки ускоряют процесс создания прототипа.

3.3 PostgreSQL

Так как наш сервис отвечает за консолидацию данных о персонале, ему необходима база данных для хранения информации о сотрудниках.

Проект использует базу данных PostgreSQL [4]. Это объектно-реляционная база данных с открытым исходным кодом, которая может быть без ограничений использована в сборке с другими программами. Из-за того, что она использует объектно-реляционную модель, в ней налажена поддержка сложных типов данных, благодаря чему становится проще выбирать данные, которые будут храниться о сотруднике.

3.4 Entity Framework Core

Для упрощения работы с данными, а также чтобы облегчить обращение к самой базе данных было принято решение использовать ORM-систему Entity Framework Core, являющуюся кросс-платформенным проектом с открытым исходным кодом [5]. Системы ORM (object-relational mapping) позволяют представить реляционную модель данных в объектно-ориентированном виде, что делает данные более понятными в использовании. Entity Framework Core также создает абстракцию над базой данных, которая упрощает к ней доступ, а также уменьшает риск совершения ошибки программистом.

3.5 Docker

Для решения проблемы развертывания продукта, состоящего из множества независимых микросервисов, было принято решение использовать контейнеризацию, которая позволяет создать портативную среду исполнения приложения и автоматизировать процесс развертывания продукта. Для этих целей была использована технология Docker [9], с помощью которой можно упаковать каждый микросервис в изолированный защищенный контейнер. Делается это посредством создания образа приложения, который содержит необходимое окружение и зависимости для запуска. Docker-контейнер является экземпляром этого образа, он работает независимо от

операционной системы хоста, благодаря чему достигается совместимость со всеми устройствами.

Кроме того, Docker — кроссплатформенный инструмент, поддерживаемый как в Linux так и в Windows. С его помощью также можно настроить систему непрерывной интеграции, которая позволит автоматизировать развертывание веб-приложения.

3.6 Nginx

В этом проекте используется прокси-сервер Nginx. Перенаправление запросов на соответствующие сервисы приложения является основной задачей прокси-сервера. Помимо этого, Nginx также может быть использован в качестве веб-сервера, который будет распределять нагрузку на приложение. Его способность обрабатывать большое число запросов одновременно, эффективно потребляя ресурсы даже при их ограниченности, позволяет ему быть использованным на разных серверах и платформах. Это делает его одним из самых популярных веб-серверов, используемых в данный момент.

3.7 Heroku

В качестве хостинговой платформы для развертывания веб-приложения был выбран PaaS-сервис Heroku [10] — среда разбработки и развертывания приложений в облаке. Помимо хостинга Heroku предоставляет набор бесплатных инструментов, таких как интегрированная прямо в облако база данных Postgres, а также удобная настройка работы в коллаборации и с GitHub, упрощающих процесс развертывания приложения. Для данного прототипа достаточно бесплатных средств Heroku, которые на других платформах, таких как AWS или Microsoft Azure бесплатно недоступны.

4. Реализация

Данный модуль посвящен описанию разработанного решения.

4.1 Описание основных компонентов архитектуры

Для данного веб-приложения была разработана микросервисная архитектура, предполагающая обособление отдельных компонентов системы в гибкие, изолированные модули небольшого размера. Каждый из таких модулей называется микросервисом и представляет собой логическую компоненту системы, отвечающую за конкретную задачу.

Поговорим подробнее об основных принципах выбранной архитектуры.

Прежде всего, каждый из микросервисов должен обладать небольшим размером. Это означает, что его реализация и поддержка не должны требовать участия целой команды людей. Архитектура микросервиса должна быть легко понимаема одним человеком, ее изменение или переписывание всего микросервиса не должно вызывать у него затруднений.

Это также обуславливается тем, что отдельный микросервис обычно имеет независимую кодовую базу, которая может быть написана на различных языках программирования с использованием специфичного стека технологий.

Также в микросервисной архитектуре характерен отказ от использования монолитной базы данных, разделенной на все части приложения. Вместо этого, каждый микросервис использует свою базу данных, благодаря чему повышается безопасность хранения данных, так как они распределяются по независимым компонентам приложения.

Принцип покомпонентного разбиения вынуждает четко определять границы микросервиса. Следствием чего является явное выделение микросервисами публичного интерфейса взаимодействия (API) [12]. Благодаря обозначенному API минимизируются зависимости между микросервисами, так как им становится доступен только ограниченный набор функций, разрешенный API конкретного компонента.

Использование микросервисной архитектуры тем самым повышает

масштабируемость приложения и упрощает внедрение в него новых компонентов, так как они могут разрабатываться независимо друг от друга на совершенно разных технологиях и языках.

На рисунке 1 представлен прототип этот архитектуры.

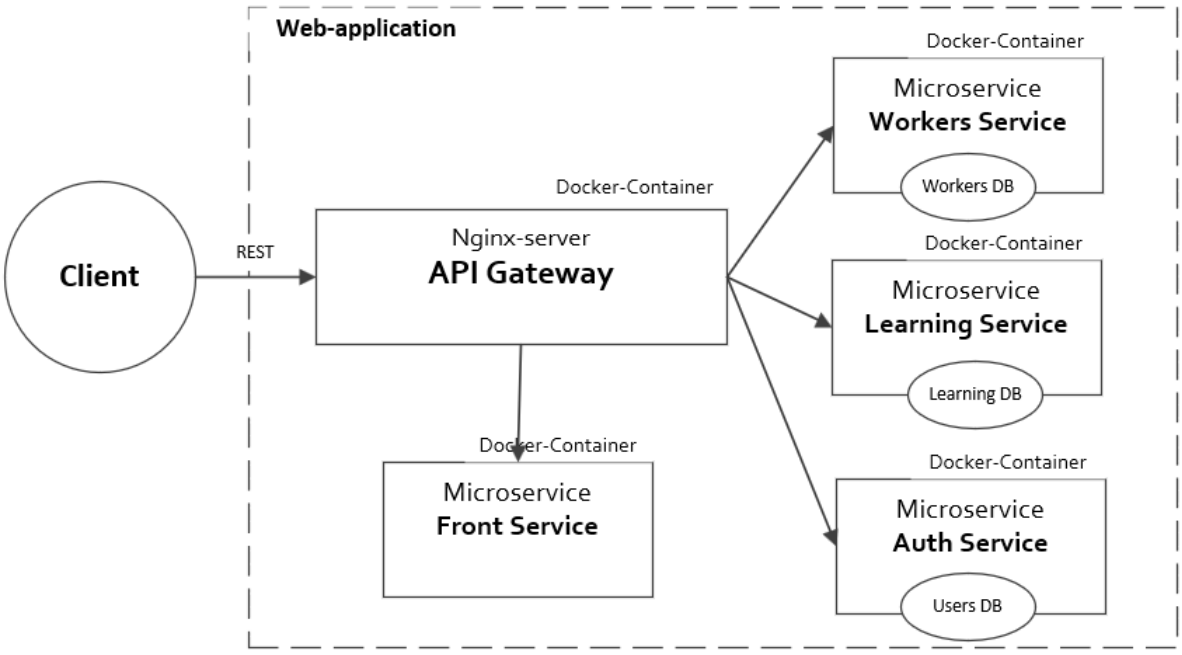

Рис. 1: Архитектура приложения.

Поступающие со стороны клиента запросы распределяются через шлюз (gateway), который перенаправляет их на нужный сервис. Общение микросервисов реализовано с помощью HTTP-запросов. Более того, в архитектуру заложен стиль REST (Representational state transfer).

Это означает, что каждый ресурс определён однозначным образом по URL-адресу. По этому адресу можно понять, какую информацию несёт в себе каждый запрос, что потенциально уменьшает ресурсоёмкость системы, так как не требуется совершать дополнительную работу на определение того, какая информация хранится в запросе.

4.2 Разработка микросервиса консолидации данных о персонале

В рамках данной работы предполагалась разработка одного из микросервисов, отвечающего за консолидацию данных о сотрудниках.

При реализации были использованы идеи архитектуры MVC (Model-View-Controller, или Модель-Вид-Контроллер), которая представляет собой покомпонентное разделение веб-приложения по следующему принципу: модель является представлением данных, которое инкапсулирует состояние приложения, вид отвечает за внешнюю составляющую приложения и запрашивает состояние модели у контроллера, с целью визуализировать эту информацию пользователю, контроллер управляет этими входящими запросами, распределяя логические задачи.

Конкретная реализация этой идеи была осуществлена с применением изменений к классическому паттерну [6]. Во-первых, вся составляющая вида была реализована отдельно на стороне клиента. Во-вторых, бизнес-логика приложения была перенесена в пакет, в котором находятся сервисы. Сервис WorkersService реализует методы добавления, удаления, а также чтения информации из базы данных. Работая с сущностями, хранящимися в пакете модели, этот сервис инкапсулирует в себе основную логику приложения. Модели - это классы, которые представляют сущности объектов базы данных: сотрудника, компетенции. Контроллеры отвечают за входящие запросы: запрос на получение всех сотрудников из базы данных, получение конкретного сотрудника, удаление, запись новых данных, передавая их сервису. Данные проходят валидацию с помощью проверки при сопоставлении моделей WorkerView и Worker. Сопоставление в свою очередь реализовано посредством встроенного класса AutoMapper, конфигурация которого лежит в пакете Mappers.

На рисунке 2 представлена диаграмма, демонстрирующая описанную архитектуру.

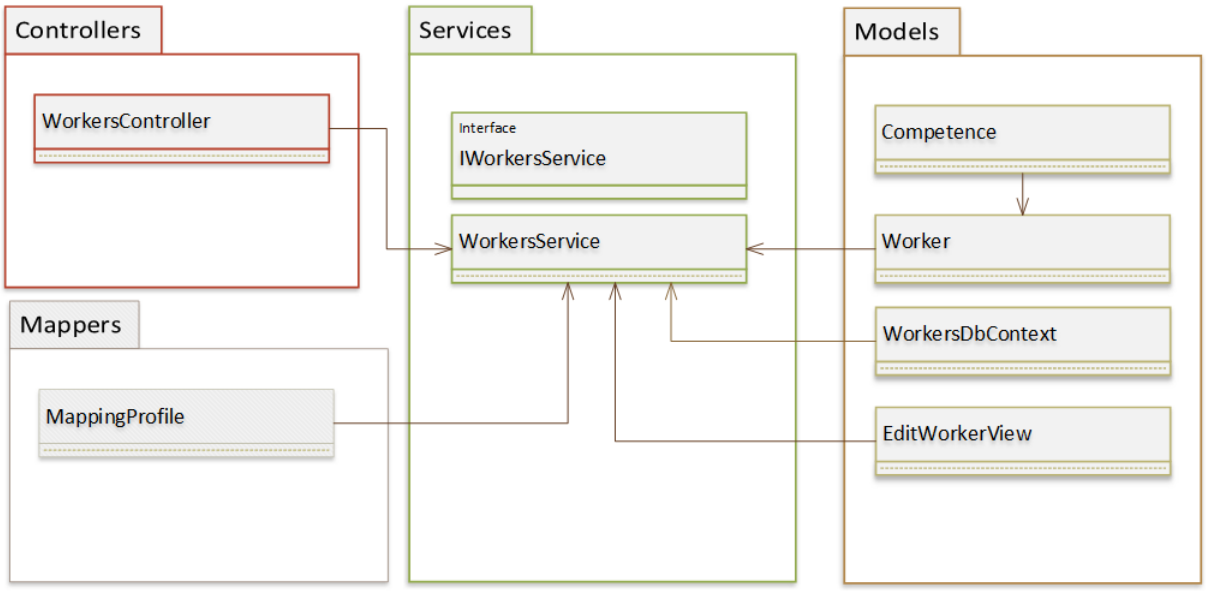

Рис. 2: Архитектура микросервиса консолидации данных о персонале.

4.2.1 Структура базы данных

База данных микросервиса реализует простую структуру, которая представлена на рисунке 3.

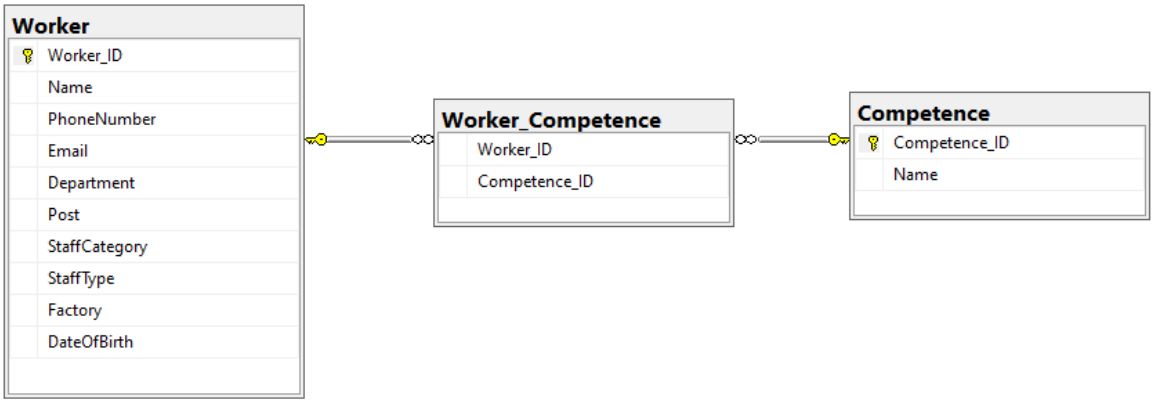

Рис. 3: Архитектура базы данных микросервиса консолидации данных о персонале.

На нём видно, что таблицы в базе данных являются репрезентацией сущностей работников (Workers) и компетенций (Competences) соответственно. Поля работника хранят в себе информацию о принадлежности к производству, категории персонала, должности, личные данные, а также связь с множеством компетенций работника.

Эти таблицы связаны между собой отношением многие ко многим, что коррелирует с идеей сопоставить работнику его множество компетенций и наоборот.

4.3 Разработка клиентской части сервиса

Перед описанием реализации интерфейса стоит поговорить о создании его первоначального макета.

Макет содержал в себе две навигационные полосы, расположенные в верхней части интерфейса. В основной же части страницы размещалась таблица, консолидирующая данные о сотрудниках, в которую были добавлены кнопки удаления и изменения содержимого. При нажатии на кнопку изменения пользователь попадал на соответствующую страницу, где мог подробнее рассмотреть информацию о сотруднике и внести туда новые данные.

Реализовано это было посредством настройки маршрутизации (роутинга). Для этого использовалась популярная библиотека React Router, которая содержит в себе полный набор компонентов, необходимых для разделения страниц.

Каждая страница представляет собой отдельный модуль, состоящий из нескольких компонентов, которые могут быть переиспользованы на других окнах веб-приложения.

Модули соединены воедино в родительском приложении, которое отвечает за маршрутизацию и организацию этих модулей. Благодаря этому для добавления нового модуля достаточно лишь импортировать его в родительское приложение. Это сделало клиентскую часть приложения легко масштабируемой.

5. Тестирование и апробация

5.1 Тестирование

Взаимодействие микросервисов было протестировано с помощью утилиты Postman [8]. Через этот сервис отправлялись соответствующие POST, GET, DELETE запросы на добавление, получение, удаление данных. В теле запроса содержалась информация об объекте сотрудника в формате JSON.

Сервер отвечал возвратом уникального номера записанного или удаленного объекта, а в случае, когда его просили выдать данные о сотрудниках или вернуть html-страницу, корректно посылал их в теле запроса.

5.2 Апробация

Перед проведением апробации продукт был развернут на веб-хостинге Heroku.

Прежде всего для каждого микросервиса было создано отдельное приложение Heroku. Предполагалось, что микросервисы будут развернуты на отдельных серверах, поэтому для каждого из них был написан dockerfile [11], который отвечал за создание образа соответствующего сервиса.

После чего был написан небольшой скрипт, который разворачивал контейнеры, содержащие микросервисы, на выделенных для них в облаке приложениях. На основном приложении, которое было доступно клиенту, размещался шлюз — прокси-сервер Nginx, перенаправляющий запросы клиента на соответствующий микросервис. Таким образом, для развертывания приложения не требовалось в ручную настраивать сервер, устанавливать на нем порты, собирать зависимости для микросервисов. Для запуска проекта была необходима лишь поддержка Docker, с помощью которого веб-приложение можно запустить как локально так и на хостинге.

Для проведения апробации были предприняты следующие шаги:

1. Базы данных приложения были заполнены синтетическими данными сотрудников, макет этих данных был получен заранее от будущих пользователей приложения.

2. Затем была составлена инструкция пользователя, с помощью которой сотрудники компании смогли бы лучше ориентироваться в использовании интерфейса веб-приложения.

3. После чего была предоставлена ссылка на основной хост приложения. Вместе с ней был приложен файл с данными для входа под несколькими видами пользователей.

4. Также пользователей попросили оценить веб-приложение по нескольким категориям по шкале от 1 до 10, где 1 — строго отрицательная оценка, 10 — строго положительная.

Приложение было протестировано сотрудниками на одном из совещаний. По итогам апробации был получен протокол совещания, в котором содержались рекомендации по улучшению сервиса.

Приложение получило следующие оценки по категориям:

- Дизайн сервиса: 8
- Удобство использования приложения: 8
- Скорость работы сайта: 10
- Полнота функционала приложения: 8
- Общая оценка сервиса: 9

Результаты

В рамках написания данной работы были получены следующие результаты:

- Проведен обзор аналогичных программных продуктов и на его основе выделены функциональные требования.
- Спроектирован и реализован микросервис консолидации данных о персонале.
- Разработан и реализован интерфейс, отвечающий за консолидацию данных о персонале
- Веб-приложение было развернуто на веб-хостинге.
- Проведена апробация среди конечных пользователей продукта.

В дальнейшем проект может продолжить развитие внутри сети компании Газпром Нефть, где он может быть встроен в кодовую базу «Цифрового двойника энергетики».

Ознакомиться с кодом проекта можно в репозитории GitHub[7].

Список литературы

[1] 1С:Зарплата и управление персоналом 8 | О продукте // 1С:Предприятие 8. — URL: https://v8.1c.ru/hrm/ (дата обращения: 12.12.2020)

[2] Microsoft Dynamics 365. Обзор управления персоналом // Microsoft. — URL: https://docs.microsoft.com/ru-ru/dynamics365/fin-ops-core/fin-ops/hr/hr-landing-page (дата обращения: 12.12.2020)

[3] Microsoft Dynamics 365. Требования к системе // Microsoft. — URL: https://docs.microsoft.com/ru-ru/dynamics365/human-resources/hr-admin-system-requirements (дата обращения: 12.12.2020)

[4] PostgreSQL: Documentation: 9.1: What is PostgreSQL? // PostgreSQL. — URL: https://www.postgresql.org/docs/9.1/intro-whatis.html (дата обращения: 29.11.2020)

[5] Microsoft Docs. Entity Framework Documentation. // Microsoft. — URL: https://docs.microsoft.com/en-us/ef/ (дата обращения: 29.11.2020)

[6] Trygve Reenskaug. The Model-View-Controller (MVC) Its Past and Present // University of Oslo. Draft of August 20, 2003 1:26 pm. / Систем. требования: Adobe Acrobat Reader. URL: http://heim.ifi.uio.no/$\sim$ *trygver*/2003/*javazone* − *jaoo*/*MVC*_*pattern.pdf* (дата обращения 14.12.2020)

[7] Репозиторий проекта на сайте GitHub // GitHub. — URL: https://github.com/Shervashidze/HRAssistantService (дата обращения: 06.06.2021)

[8] Postman. How to Use Postman for API Testing Automation // Postman. — URL: https://www.postman.com/use-cases/api-testing-automation/ (дата обращения: 23.05.2021)

[9] Merkel D. Docker: lightweight linux containers for consistent development and deployment. // Linux journal. — 2014. — 239 — P. 2.

[10] About Heroku. Официальная страница Heroku. // Heroku. — URL: https://www.heroku.com/about (дата обращения: 06.06.2021)

[11] Официальная документация Docker. Dockerize an ASP.NET Core application. // Docker. — URL: https://docs.docker.com/samples/dotnetcore/ (дата обращения: 06.06.2021)

[12] Wikipedia. API management. // Wikipedia, the free encyclopedia. — URL: https://en.wikipedia.org/wiki/API_management (дата обращения: 06.06.2021)