# Аннотация

Пояснительная записка к выпускной квалификационной работе на тему «Инструментальный комплекс создания программ для обучения английскому языку отдельных категорий пользователей. Серверная часть» представлена на 61 странице, содержит 31 рисунок, 6 таблиц.

Первая глава состоит из трех параграфов. В первом параграфе описываются анализ состояния рассматриваемой области, во второй уточнение требований к разрабатываемой системе, представляется ТЗ на серверную часть инструментального комплекса.

Вторая глава состоит из четырех параграфов. В первом параграфе выбор инструмента проектирование серверной части, во втором анализ и выбор архитектуры системы, в третьем проектирование информационных модулей, в четвертом приводятся алгоритмы.

Третья глава состоит из трех параграфов. В первом параграфе обзор технологий, используемых сервером, во втором обзор инструментов разработки, в третьем описание используемых паттернов, в четвертом описание реализации базы данных.

Четвертая глава – тестирование, состоит из двух параграфов. В первом параграфе описывается тестирование, во втором описывается автоматическое тестирование.

Пятая глава состоит из двух параграфов. В первом параграфе описывается организация и планирование работ по теме, во втором расчет стоимости проведения работ.

**Оглавление**

[Аннотация 1](#_Toc42503757)

[Список используемых сокращений 3](#_Toc42503758)

[Введение 4](#_Toc42503759)

[1 Аналитический раздел 6](#_Toc42503760)

[1.1 Анализ состояния рассматриваемой области 6](#_Toc42503761)

[1.2 Уточнение требований к разрабатываемой системе 7](#_Toc42503762)

[1.3 ТЗ на серверную часть инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей. 10](#_Toc42503763)

[2 Проектный раздел 14](#_Toc42503764)

[2.1 Выбор инструмента проектирования серверной части 14](#_Toc42503765)

[2.2 Анализ и выбор архитектуры системы 15](#_Toc42503766)

[2.3 Проектирование информационных модулей создаваемого решения 18](#_Toc42503767)

[2.4 Моделирование бизнес-процессов предметной области (Алгоритмы) 23](#_Toc42503768)

[3 Технологический раздел 29](#_Toc42503769)

[3.1 Обзор технологий используемых сервером ИК 29](#_Toc42503770)

[3.2 Обзор инструментов разработки 32](#_Toc42503771)

[3.3 Описание используемых паттернов 41](#_Toc42503772)

[3.4 Описание реализации базы данные 44](#_Toc42503773)

[4 Тестирование 45](#_Toc42503774)

[4.1 Описание тестирования 45](#_Toc42503775)

[4.2 Описание непрерывной интеграции 48](#_Toc42503776)

[5 Экономический раздел 51](#_Toc42503777)

[5.1 Организация и планирование работ по теме 51](#_Toc42503778)

[5.2 Расчёт стоимости проведения работ 54](#_Toc42503779)

[Заключение 60](#_Toc42503780)

[Список литературы 61](#_Toc42503781)

[Приложение 1 – Код непрерывной интеграции 64](#_Toc42503782)

# Список используемых сокращений

**REST** – это архитектурный стиль распределенных систем, используется для проектирования веб-служб.

**gRPC** – это высокопроизводительный RPC-фреймворк с открытым исходным кодом, который может работать в любой среде. Он может эффективно подключать службы в центрах обработки данных и между ними с подключаемой поддержкой балансировки нагрузки, трассировки, проверки работоспособности и аутентификации. Он также применим в сетях с низким качеством связи, для подключения устройств, мобильных приложений и браузеров к серверным службам API.

**CI/CD** (Continuous integration & Continuous delivery) — методология разработки программного обеспечения, особенность которой — непрерывность процесса, реализуемая с помощью различных систем управления Agile

**SDK** (от англ. software development kit) — набор средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определённого пакета программ.

**JWT** – JSON Web Token, объект стандарта RFC 7519. Считается безопасным способом обмена информации между участниками. Состоит из трех частей: заголовка, полезной нагрузки и подписи.

**CDN**, Content Distribution Network — географически распределённая сетевая инфраструктура, позволяющая оптимизировать доставку и дистрибуцию содержимого конечным пользователям в сети Интернет.

# Введение

Английский – это наиболее часто используемый язык во всём мире. Многие международные компании проводят встречи на английском языке, университеты преподают курсы на английском языке, туристы и путешественники используют английский язык в качестве общего языка, также знание английского языка позволит эффективнее находить информацию в сети интернет.

Текущий уровень развития информационных технологий позволяет автоматизировать процессы обучения. Тенденция внедрения электронного обучения наблюдается повсеместно, а главными инструментами познания в век информации стали различные виды программного обеспечения.

В работе представлены этапы создания серверной части инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей.

Подавляющее большинство систем основаны на клиент-серверной архитектуре, также известной как сетевая вычислительная модель или клиент-серверная сеть, поскольку все запросы и службы доставляются по сети. [1]

Стремительное увеличение производительности процессоров, согласно закону Мура, остаётся в силе даже после того, как в 2007 году Мур заявил, что закон, очевидно, скоро перестанет действовать [2]. Уже на данных момент процессоры настольных компьютеров и мобильных телефонов позволяют запускать сложные приложения. Тем не менее, серверная часть остаётся лучшим решением для обеспечения безопасности, в виду открытости кода на веб приложениях, и синхронизации данных между устройствами.

Целью выпускной квалификационной работы является обеспечение эффективной поддержки работы мобильных и веб-клиентов путем вынесения логики работы с данными на серверную часть. Для достижения поставленной цели решаются следующие задачи:

1. Реализация непрерывной интеграции и развертывания с использованием сервиса Azure DevOps Pipeline для повышения скорости поставки продукта конечным пользователям.
2. Обеспечивание журналирования и метрики для обеспечения своевременного получения информации о неполадках и необходимости оптимизировать методы. Также определения работоспособности программного продукта после его публикации.
3. Настройка автоматического тестирования с использованием сервиса Azure DevOps Tests для повышения качества поставляемого продукта
4. Реализация автоматического документирования по спецификации OpenAPI для быстрой синхронизации текущего функционала и его описания, а также для экономии времени на разработку.
5. Аналитический раздел
   1. Анализ состояния рассматриваемой области

Английский – это наиболее часто используемый язык во всём мире. Многие международные компании проводят встречи на английском языке, университеты преподают курсы на английском языке, туристы и путешественники используют английский язык в качестве общего языка, также знание английского языка позволит эффективнее находить информацию в сети интернет.

Несмотря на то, что в рассматриваемой предметной области относительно много общедоступных продуктов, только малая часть, обладающая схожим функционалом, является программным обеспечением с открытым исходным кодом и абсолютное меньшинство систем заботятся о поддержке мобильных клиентов.



Рисунок 1 – Аналоги

Среди рассмотренных наиболее схожих приложений, ни одно из них полностью не обладает одновременно всеми свойствами, реализованными в текущей ВКР. Приложения имеющие готовую базу материала типа: Linqualeo, Duolinquo являются проприоритарными и не имеют открытого API. С другой стороны приложения с открытым исходным кодом типа Wordsteps и Anki не хотя и имеют открытый API, не обладают готовой базой материала, а так же подключение через REST поддерживается не разработчиком а сообществом, из-за чего обновления поступают с задержкой. Также не одно из приложений не обеспечивает открытое подключение через gRPC для обеспечения производительного обмена данными с мобильными устройствами.

Таким образом, разработка системы обучения английскому языку «Инструментальный комплекс создания программ для обучения английскому языку отдельных категорий пользователей. Серверная часть», обладающая свойствами: наличие открытого API, REST подключение, GRPC подключение, готовая база материала – является актуальной задачей.

* 1. Уточнение требований к разрабатываемой системе

Подавляющее большинство систем основаны на клиент-серверной архитектуре, также известной как сетевая вычислительная модель или клиент-серверная сеть, поскольку все запросы и службы доставляются по сети. [1]

Конечными клиенты являются устройства разного типа, подключенные к сети на различных операционных системах. При этом особое внимание должно уделяться возможности подключения мобильных устройств с малой пропускной способности сети.

Ставится задача создания серверной части инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей.

К серверной части инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей предъявляются такие общие требования как:

1. система должна поддерживать постоянную интеграцию и поставку;
2. сервер должен находиться на территории РФ;
3. минимизация размера передаваемых данных;
4. открытый программный код.

Первое требование базируется на использовании для разработки стиля гибкой разработки. С учетом применения разработки в стиле agile, система должна поддерживать постоянную интеграцию и поставку. Agile (Гибкая разработка) от слова гибкость, как способность изменяться, быстро реагировать на изменения. Способ справиться с неопределенной и неспокойной средой и в конечном итоге добиться желаемого результата. Гибкая разработка программного обеспечения — это больше, чем такие практики, как парное программирование, разработка на основе тестов, планирование сессий и спринтов. Гибкая разработка программного обеспечения – это обобщающий термин для набора практик, основанных на ценностях и принципах, выраженных в манифесте гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе. Из них два говорят о том, что работающий продукт нужно выпускать как можно чаще, с периодичностью от пары недель до пары месяцев и о том, что работающий продукт – основной показатель прогресса [3]. Для того, чтобы реализовать быстрый выпуск продукта, нужно реализовать CI/CD конвейер. Для повышения надежности системы нужно обеспечить значительное покрытие тестами и проводить частое автоматическое тестирование. Автоматическое тестирование так-же необходимо из-за частых поставок продукта, чтобы сократить время ручного тестирования. CI/CD как правило предусматривают сервисы автоматического тестирования как часть конвейера непрерывной интеграции.

Второе требование учитывает закон об «Обработке и хранении персональных данных в РФ» от 1 сентября 2015 года, все персональные пользовательские данные должны храниться на территории Российской федерации;

Третье требование основывается на том, что основные пользователи сети интернет – мобильные устройства [4], нужно минимизировать размер передаваемых данных, чтобы ускорить работу конечных клиентов.

Рисунок 2 – рыночная доля устройств

Таким образом из рисунка 1 видно, идет тенденция к увеличению доли мобильных устройств и поэтому требование по уменьшению объема передаваемых данных по сети является практически обязательным для современных серверов.

В соответствии с четвертым требованием разработанная система должна быть программным продуктом с открытым исходным кодом системах. Система с открытым исходным кодом внушает большее доверие, мотивирует разработчиков писать чистый код, дает потенциально более пригодно для создания сообщества разработчиков вокруг программного продукта. Также серверное приложение и запускаться на Unix-подобных операционных, с учетом того, что публичные серверы сети интернет в большинстве своем (70.8% на май 2020) используют операционные системы на базе ядра Linux [5].

Таким образом сформулируем задачу разработки серверной части инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей базирующуюся на основных четырех требованиях.

* 1. ТЗ на серверную часть инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей.

**Общие сведения**

Наименование системы. Обучающая программа тренажерного типа для обучения английскому языку отдельных категорий пользователей «InWords». Серверная часть. Условное наименование системы **–** «InWords»

В соответствии с Положением об итоговой государственной аттестации выпускников высших учебных заведений в Российской Федерации, утвержденным постановлением Государственного комитета РФ по высшему образованию от 25 мая 1994 г. №3 защита выпускной квалификационной работы является обязательной составляющей итоговой государственной аттестации выпускников вузов. Разработка ведется на основании задания на ВКР утвержденного РТУ МИРЭА 27 апреля 2020 г.

Начало работ – сентябрь 2019 г. Окончание работ – апрель 2020 г.

Система предназначена для автоматизации процесса изучения иностранной лексики с сохранением прогресса для каждого пользователя.

Целью создания Системы является расширения словарного запаса английского языка среди отдельных категорий пользователей.

Объектом автоматизации является процесс изучения иностранной лексики.

Процесс изучения лексики состоит из двух частей:

1. Подбор слов, которые следует изучать
2. Запоминанием слов и их перевода.

Основные проблемы, с которыми можно столкнуться:

1. Неправильные выбор слов для изучения (изучение редко используемых слов может стать пустой тратой времени).
2. Некачественное запоминание слова и его перевода (без регулярного повторения слово быстро забывается).

Первая проблему решается предоставлением пользователю слов из словаря частотности английского языка. Вторая проблема решается использованием напоминаний через определенные промежутки времени о необходимости повторения слов.

Требования к системе. Система может быть быть распределенной, но все данные, имеющие отношение к пользователям, должны располагаться в едином хранилище на территории РФ.

Основные части системы:

* Сервер Приложений. Обрабатывает все данные и предоставляет доступ к ним через открытый API.
* Сервер управления. Содержит панель управления для мониторинга состояния и упрощения администрирования и обслуживания.
* FTP Доступ для автоматизации развертывания Сервера приложений
* Файловый Сервер. Содержит дополнительные данные сервера приложений, такие как шаблоны писем, файлы журнала работы, настройки, файлы установки ПО.
* Сервер базы данных. Содержит пользовательскую информацию
* Сервер резервных копий. Хранит копию всех серверов за последние 3 дня, делается ежедневно.
* Прокси сервер переадресовывает и фильтрует все входящие запросы.

Для работы программы необходимо как минимум два человека – системный администратор и оператор. Каждый из них должен иметь высшее профильное образование.

Информационная система должна восстанавливать свою работу после сбоев электропитания или перезагрузки без участия человека. Кроме случаев повреждения носителей информации с исполняемым кодом.

Необходимо, чтобы ИС могла обеспечить верную обработку аварийных ситуаций, вызванных при использовании неверных форматов передачи данных или недопустимыми входными данными.

Подсистема авторизации пользователей должна обеспечивать защищенный сеанс установки соединения клиента и сервера с использованием JSON Web Token (JWT).

Для эксплуатации разрабатываемой информационной системы необходимы следующие условия:

1. Сеть высокопроизводительного взаимодействия вычислительных средств со скоростью обмена данными между конечными узлами серверного комплекса сети не менее 100 Мбит/сек (Fast Ethernet);

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации;

Подсистема авторизации пользователей

Функции:

* Регистрация пользователя в системе;
* Аутентификация и авторизация пользователя.

Подсистема обработки запросов пользователей

Функции:

* Добавление пользователя;
* Удаление пользователя;
* Поиск заданного пользователя;
* Добавление словаря пользователя;
* Удаление словаря пользователя;
* Поиск словаря пользователя;
* Редактирование словаря;
* Сохранение результата тренировки.

Подсистемы взаимодействия с пользователем

Функции:

* Загрузка словаря;
* Просмотр словаря;
* Редактирование словаря в локальном репозитории;
* Передача изменений базе данных;
* Выполнение интерактивных заданий.

Требования к математическому обеспечению

Для обеспечения безопасности требуется задействовать алгоритм симметричного шифрования токена авторизации и Rfc2898 шифрование паролей.

Для расчета интервала повторения требуется задействовать математическую формулу интервального повторения, выведенную по экспериментам Эббингауза или любой другой альтернативный закон интервальных повторений.

Информационная совместимость подсистем клиента и сервера должна обеспечивается на уровне обмена JSON объектами по протоколу HTTPS/1, HTTPS/1.1, HTTPS/2 и посредством GRPC по протоколу HTTP/2.

Программа написана с использованием языков программирования платформы .NET (C#, F#)

Веб интерфейс программы должен содержать страницы на английском или русском языках.

Программное обеспечение серверной части ИС: CentOS 7, СУБД Mysql MariaDB 10.X.

Программно-технические средства компонентов системы должны соответствовать стандартам обмена с использованием протокола TCP/IP.

Требования к техническому обеспечению

- Процессор: от 2900 МГц, 2 ядра;

- Оперативная память: от 1.5 Гб и более.

- Диск, SSD RAID 10: от 10 Гб

- Резервное хранилище от 20 Гб

Требования к методическому обеспечению

* Руководство администратора
* Руководство разработчика

Также методическое обеспечение должно быть достаточным для выполнения всех функций серверной части Инструментальный комплекс создания программ для обучения английскому языку отдельных категорий пользователей.

Таким образом В параграфе было разработано техническим задание в соответствии с стандартом ГОСТ 34.602-89.

Выводы.

В главе аналитический обзор были:

* проведен анализ систем обучения английскому языку, обладающих свойствами: открытое API, REST подключение, gRPC подключение, готовая база материала;
* сформулирована задача разработки серверной части инструментального комплекса создания программ для обучения английскому языку отдельных категорий пользователей базирующуюся на основных четырех требованиях – открытое API, REST подключение, gRPC подключение, готовая база материала;
* показана актуальность задачи разработки серверной части инструментального комплекса;
* разработано техническое задание.

1. Проектный раздел
   1. Выбор инструмента проектирования серверной части

На основе проведенного анализа в качестве основного инструмента проектирования был выбран Draw.io – бесплатный онлайн инструмент, который позволяет создавать схемы любой сложности с большим количеством встроенных готовых макетов, включающих:

Макет диаграммы классов

ER-диаграммы

Блок-схемы

ITIL Архитектура системы

Шаблон сетевого отображения

Помимо неформальных схем, draw.io позволяет проектировать в стандартных нотациях таких как UML. DFD, IDEF0, IDEF1X, BPMN, диаграммы деятельности и другие.

Draw.io входит в компонент G SUIT, это означает что данный инструмент зачастую входит в стандартные инструменты проектирования на предприятиях. Помимо этого, draw.io поддерживается и разрабатывается компанией Google, что обеспечивает продукту длительную поддержку и хорошую документацию.

Кроме наличия большого набора шаблонов draw.io обеспечивает совместный доступ, быстрое моделирование, интуитивно понятный интерфейс, экспорт проекта в векторные и растровые форматы.

Несмотря на то, что изначально draw.io это онлайн сервис, существует официальное приложение доступное для персонального компьютера, установив которое можно получить доступ к сервису без использования интернета.

* 1. Анализ и выбор архитектуры системы

Согласно требованиям, представленным в параграфе 1.2, серверная часть ИК должна поддерживать соединение с мобильными и веб клиентами. Для этого проводится сравнение архитектуры клиент-сервер и одноранговой архитектуры.

Архитектура клиент-сервер

Архитектура клиент-сервера — это вычислительная модель, в которой сервер размещает, поставляет и управляет большинством ресурсов и услуг, потребляемых клиентом. Этот тип архитектуры имеет один или несколько клиентских компьютеров, подключенных к центральному серверу по сети или интернету. Эта система совместно использует вычислительные ресурсы. Архитектура клиент-сервер также известна как сетевая вычислительная модель или клиент-серверная сеть, поскольку все запросы и службы доставляются по сети.

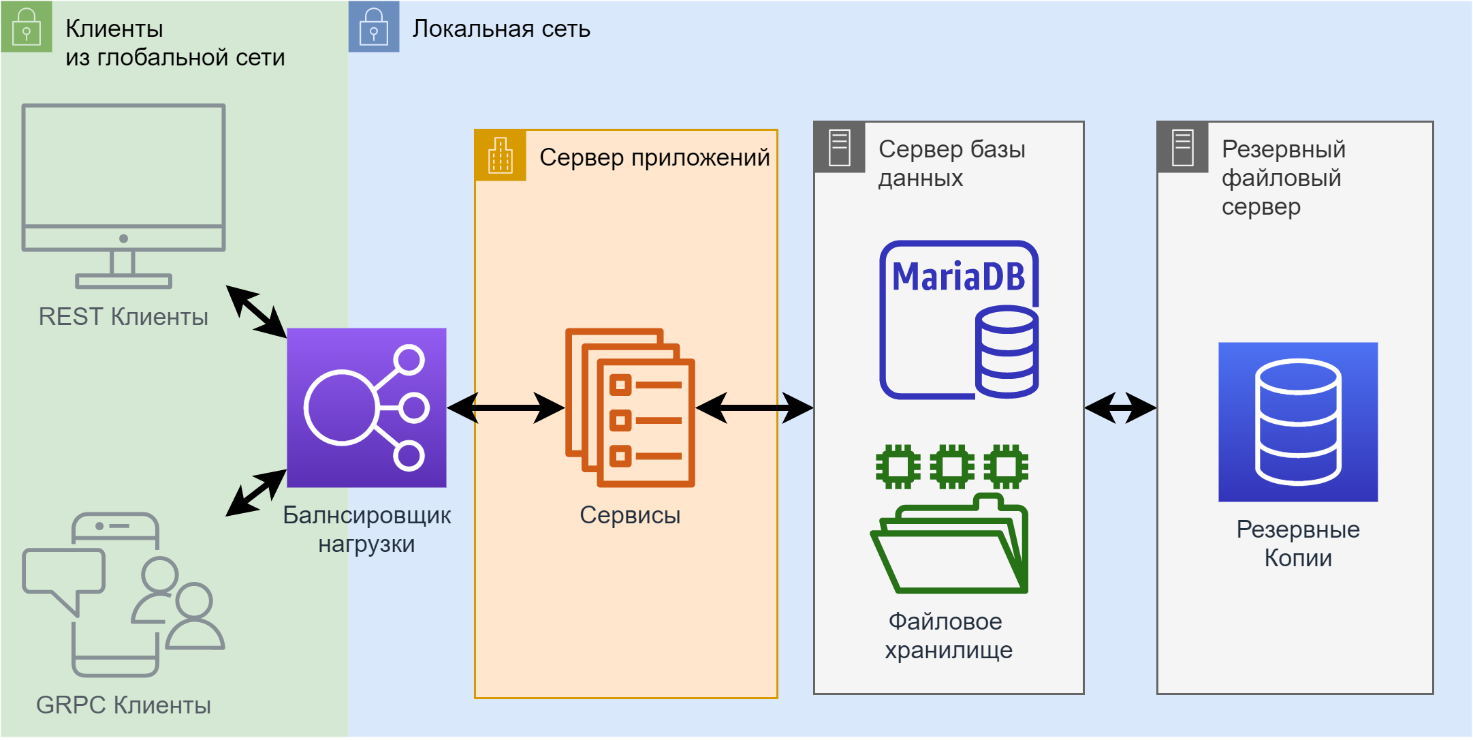


Рисунок 3 – Принцип работы комплекса

Традиционная архитектура клиент-сервер включает в себя два уровня: уровень клиента и уровень сервера. Еще одна распространенная конструкция клиент-серверных систем использует три уровня:

1. Клиент, который взаимодействует с пользователем (Пользовательски интерфейс)
2. Сервер приложений, содержащий бизнес-логику приложения (Приложение)
3. Менеджер ресурсов, хранящий данные (База данных)

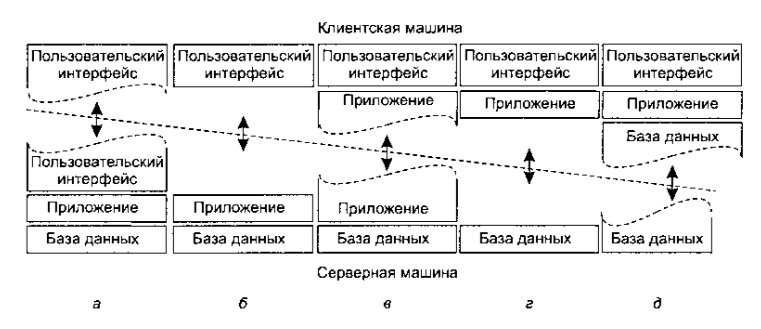


Рисунок 4 – Типы клиентов

**Одноранговая архитектуры**

Главное различие одноранговых и клиент-серверных систем в том, что в первой, в узлы выступают как поставщики услуг, так и потребители услуг, а во второй клиенты, запрашивающие услуги, а серверы их предоставляют. Кроме того, клиент-серверные системы требуют центрального сервера, и их создание обходится куда дороже, чем одноранговые системы. Однако в системе клиент-сервер выделенный центральный сервер обеспечивает больший уровень безопасности, чем одноранговые системы, в которой безопасность обрабатывается клиентами.

Одноранговые сети испытывают проблемы производительности по мере увеличения числа узлов, клиент-серверные системы более стабильны и могут масштабироваться столько, сколько вам нужно. Поэтому выбор зависит от среды, которую необходимо реализовать.

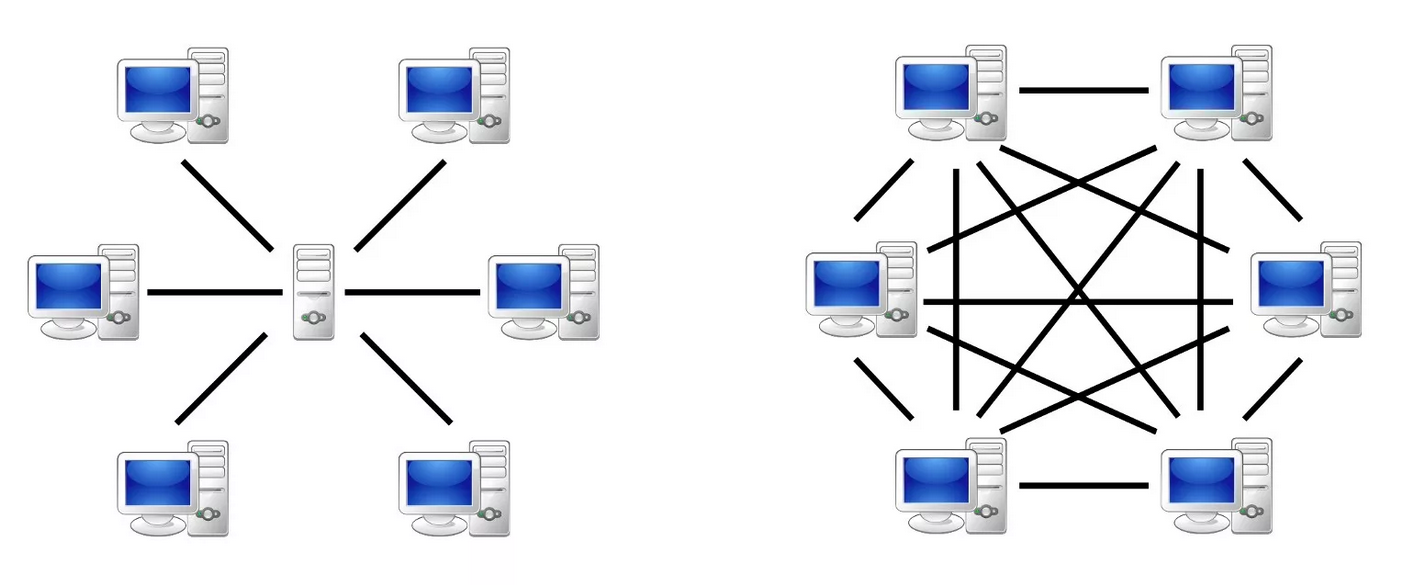


Рисунок 5 –Модель архитектуры клиент-сервер двух различных типов.

В распределенной архитектуре (слева) одна или несколько выделенных машин используются в качестве сервера, в то время как все остальные машины используются в качестве клиентов. В этом случае клиенты могут взаимодействовать через сервер.

В одноранговой архитектуре (справа) каждый узел или экземпляр прикладной программы может функционировать одновременно как клиент и сервер. Оба они имеют одинаковые обязанности и статус.

Одноранговая сеть не пригодна для решаемой задачи из-за получаемой избыточности информации, находящейся на клиенте и сложности распространения обновлений.

По результатам сравнения клиент-серверная архитектура является более целесообразной для рассматриваемой задачи, из-за простоты подключения клиентов к серверу.

* 1. Проектирование информационных модулей создаваемого решения
     1. Обеспечение безопасности

Для обеспечения безопасности каждый пользователь подключается и использованием JWT. JSON Web Token, объект стандарта RFC 7519. Считается безопасным способом обмена информации между участниками. Состоит из трех частей: заголовка, полезной нагрузки и подписи. В качестве шифрование используется симметричное шифрование ключ которого лежит на сервере, никто не имеет к нему доступ. Симметричное шифрование в данном случае лучше ассиметричного из-за того, что ключ не покидает места где он используется. В качестве объекта запроса токена используются стандартные поля аутентификации: email и пароль или же аунтефикация через google.

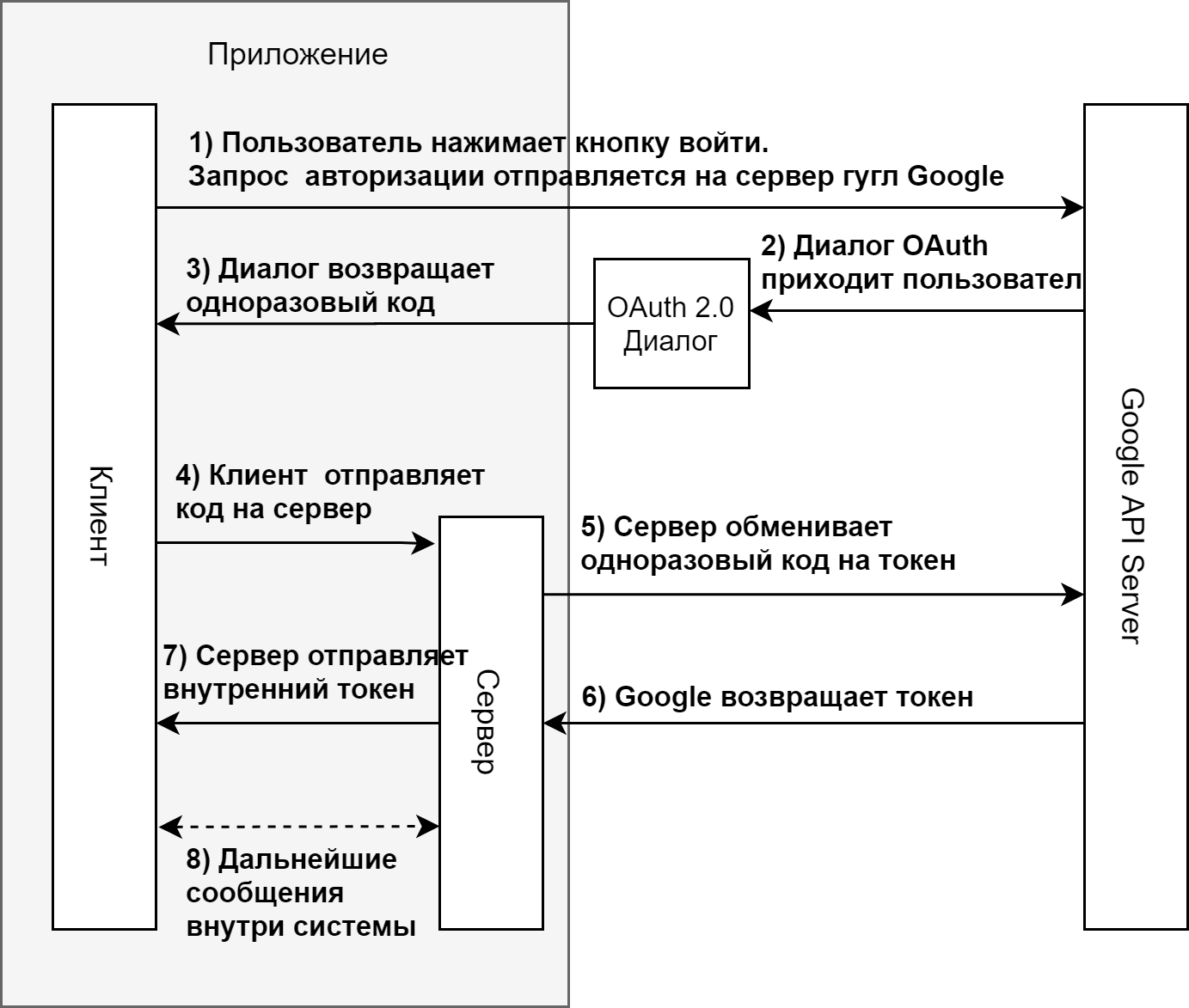


Рисунок 6 – Вход в Google для серверных приложений

В случаи с авторизацией через Google для предотвращения постоянной связи с google и сохранения пользовательских данных внутри приложения используется модель аутентификации, представленная на рисунке 9.

Рабочий процесс, описанный Google является гибридным, когда пользователь авторизует приложение на стороне клиента, отправляется специальный одноразовый код авторизации на свой сервер. Сервер обменивает одноразовый код для получения собственных токенов доступа, чтобы в дальнейшим клиент мог использовать API сервера без запросов в сторону Google. [10]

Кроме безопасности передачи данных пользователей путем защиты сессии с помощью JWT токена применен метод шифрования пароля с солью.

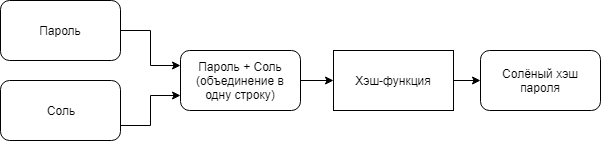


Рисунок 7 – Шифрование с солью

Соль (также модификатор входа хэш-функции) — строка данных, которая передаётся хеш-функции вместе с входным массивом данных (прообразом) для вычисления хэша (образа). [11] Существует несколько способов хранить пароли в базе данных. Первый – хранить пароль в открытом виде, в случаи взлома базы данных, пароль пользователя достаётся злоумышленникам, по статистике пользователь использует один и тот же пароль для разных ресурсов и при потере своего пароля пользователь отдает все свои сервисы злоумышленникам. Второй способ — это использование чистой хэш функции. При утечки значения хэш функции пароль не может быть восстановлен и злоумышленник не получает доступа ни к сервису где произошла утечка базы данных ни к другим сервисам в которых зарегистрирован пользователя, однако существует метод взлома таких паролей с использованием «радужных таблиц» злоумышленники перебирают все значения и прогоняют их через популярные хэш функции, эта таблица постоянно накапливается и даже существуют бесплатные сервисы в интернете которые восстанавливают пароль по хэшу (например MD5). Третий способ — это Хеширование «солёного» пароля, такой метод призван предотвратить возможность восстановить пароль при помощи радужных таблиц. Поскольку один и тот же пароль, зарегистрированный разными пользователями, выдаст разные хэш функции. «Соли» добавляют столько, чтобы пароль был значительного размера и был криптостойким.

Для использования на сервере инструментального комплекса был выбран третий способ шифрования паролей а именно хеширование с солью.

* + 1. Индивидуальный пользовательский словарь

Словарь пользователя наполняется по мере прохождения им курсов или же по вручную самим пользователем. Каждое слово имеет поле уникального ключа, строковое поле для слова на родном языке, строковое поля для слова на иностранном языке, номер словаря пользователя, идентификатор пользователя, период изучения и временной промежуток до следящего повторения. Алгоритм подбора слов будет рассмотрен в пункте 2.3.1

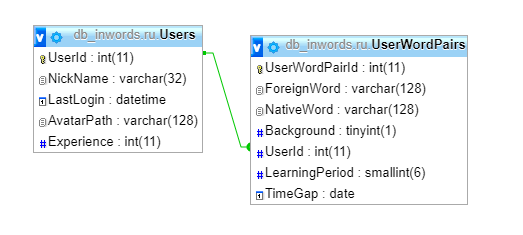


Рисунок 8 – частичная схема таблицы пользовательских слов

Сервер предоставляет методы синхронизации словарей который так же будет рассмотрен главе про алгоритмы 2.3.4.

Пользовательский словарь предоставляет следующий функционал для каждого серверной части ИК:

* Добавление слов в словарь
* Редактирование слов словаря
* Перевод нового добавленного слова
* Получение слов для интервального повторения
* Получение ключей слов для тренировки
* Удаление слов из словаря

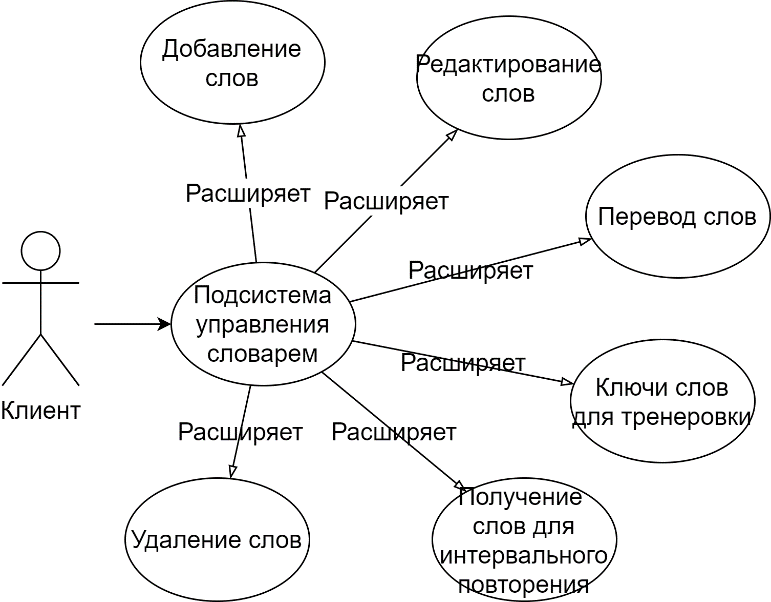


Рисунок 9 – варианты использования словаря

* + 1. Общие наборы слов

Наборы слов – это готовые, популярные слова, предлагаемые пользователю на изучение. состоят из уровней.

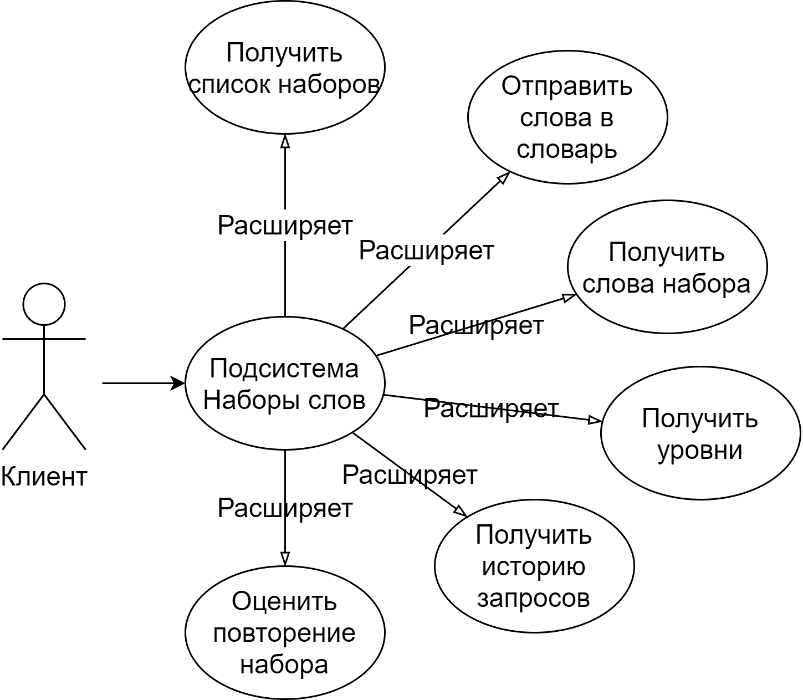


Рисунок 10 – Варианты использования набор слов

Основной функционал заключается в том, чтобы предоставить клиенту информацию об официальных и пользовательских наборах слов, а также запросить оценку о текущем уровне запоминания слов в наборе.

В параграфе были представлены 3 модуля – безопасности, пользовательского словаря и общего набора слов.

* 1. Моделирование бизнес-процессов предметной области (Алгоритмы)
     1. Алгоритм подбора слов

В основе алгоритмы подбора слов лежат эксперименты немецкого ученого Эббингауза. «Кривая забывания» или «Кривая Эббингауза» была получена вследствие экспериментального изучения памяти немецким психологом Германом Эббингаузом в 1885 году. [12]

С 1880 по 1885 год Герман Эббингауз провел ограниченное, неполное исследование над собой и опубликовал гипотезу в 1885 году под названием "Über das Gedächtnis" (позже переведенное на английский язык как "память: вклад в экспериментальную психологию"). Эббингауз изучал запоминание бессмысленных слогов, таких как "WID”, "ZOF", “CVC” или согласный–гласный–согласный), многократно проверяя себя через различные промежутки времени и записывая результаты. Он построил эти результаты на графике, создав то, что теперь известно как "кривая забвения".[13]

Публикация Эббингауза также включала уравнение для аппроксимации его кривой забывания [14]:

Где **b** уровень «сохранения» выражается в процентах, и **Т**-время в минутах. Уровень сохранения определяется как относительное количество времени, сэкономленное на втором испытании в результате проведения первого. Экономия в размере 100% будет означать, что все объекты (слова) ещё были известны (запомнены) с первого испытания. Уровень сохранения памяти 75% означала бы, что повторное изучение пропущенных объектов потребовало бы 25% времени, необходимого для первоначальной сессии обучения (чтобы выучить все объекты). Таким образом, уровень запоминания аналогичен степени удержания знаний в памяти.

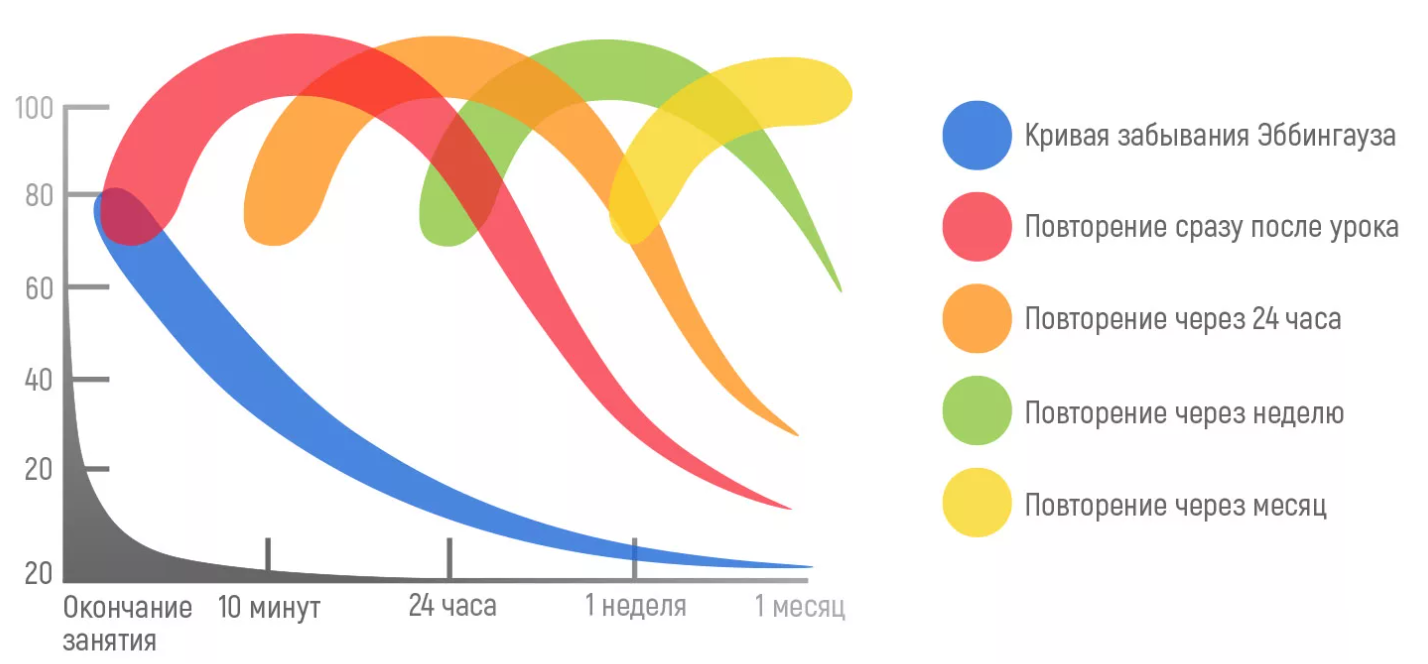


Рисунок 11 – Интервальные повторения

После экспериментов Эббингауза было ещё много исследований и предложенных уравнений «забывания» возможно, одним из которых является экспоненциальная кривая, описываемая экспоненциальным уравнением

Где R – легкость повторения (коэффициент того насколько хорошо информация осталась в памяти) S – стабильность памяти, определяет, насколько быстро R уменьшается спустя время после последней тренировки и t – время соответственно.

Простые уравнения, подобные этому, были найдены Рубином, Хинтоном и Венцелем (1999), чтобы обеспечить хорошее соответствие имеющимся данным [14]

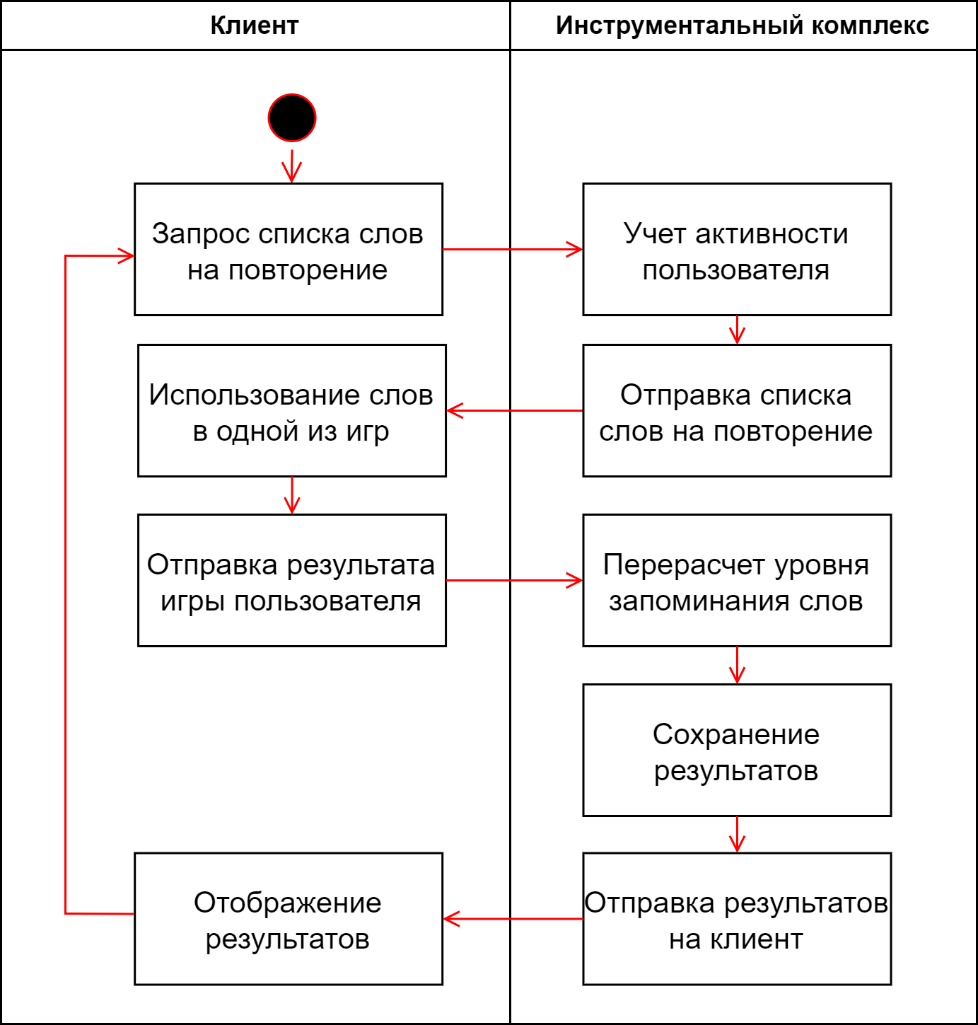


Рисунок 12 – Алгоритм подбора слов

В результате проведенного анализа был построен алгоритм подбора слов представленный на рисунке 12.

* + 1. Алгоритм оценки уровня игры

Суть уровня игры заключается в открытии закрытых карточек путём попарного сопоставления слова с переводом.

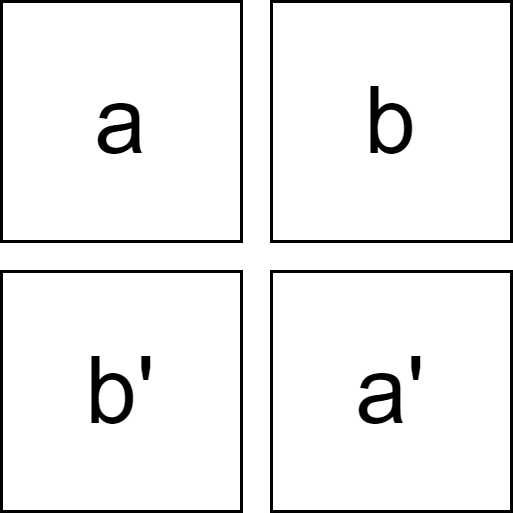


Рисунок 13 – Упрощенный пример игры

Количество карточек может быть любым четным числом, предполагается что самый легкий уровень содержит в себе как минимум 2 пары слов (4 карточки) а самый сложный 8 пар слов (16 карточек). Изначально пользователь не видит, что располагается на карточке и открывает их одна за другой, при открытии первой карточки пользователь откроет то верным решением будет считаться открытие пользователем карточки со значением . В общем случаи для того, чтобы пользователь открыл все карточки нужно затратить шагов, n, где – число карточек. Именно потому что будем считать, что пользователю «навезёт» и он всегда открывает пару, которая не подходит друг к другу. Тем не менее последние две всегда можно вычислить путём исключения, например если пользователь открыл сначала совершив 2 шага, далее открыв пользователь уже знает где открывал b и далее открыв уже знает где a. Таким образом пользователь за 6 отрытый {(a;b),(b’;b),(a’;a)0}при полном «невезении» может успешно пройти уровень на максимальную оценку. Уровни оцениваются от 0 до 6 условных единиц, где 6 наивысшая оценка. Оценка изменяется в зависимости от допущенных ошибок и распределяется по следующему формуле, полученной эмпирическим путём.

Где **y** это итоговая оценка, **x** количество открытий карточек, **b** количество открытий, считающихся наилучшим исходом.

* + 1. Алгоритм метрики

Разработано два типа метрики. Первый тип учитывает время доставки ответа на клиент. Такая метрика полезна для вычисления времени отправления больших данных или клиентов, расположенных далеко от сервера. В будущем с помощью этой метрики можно будет оценивать качество работы CDN.

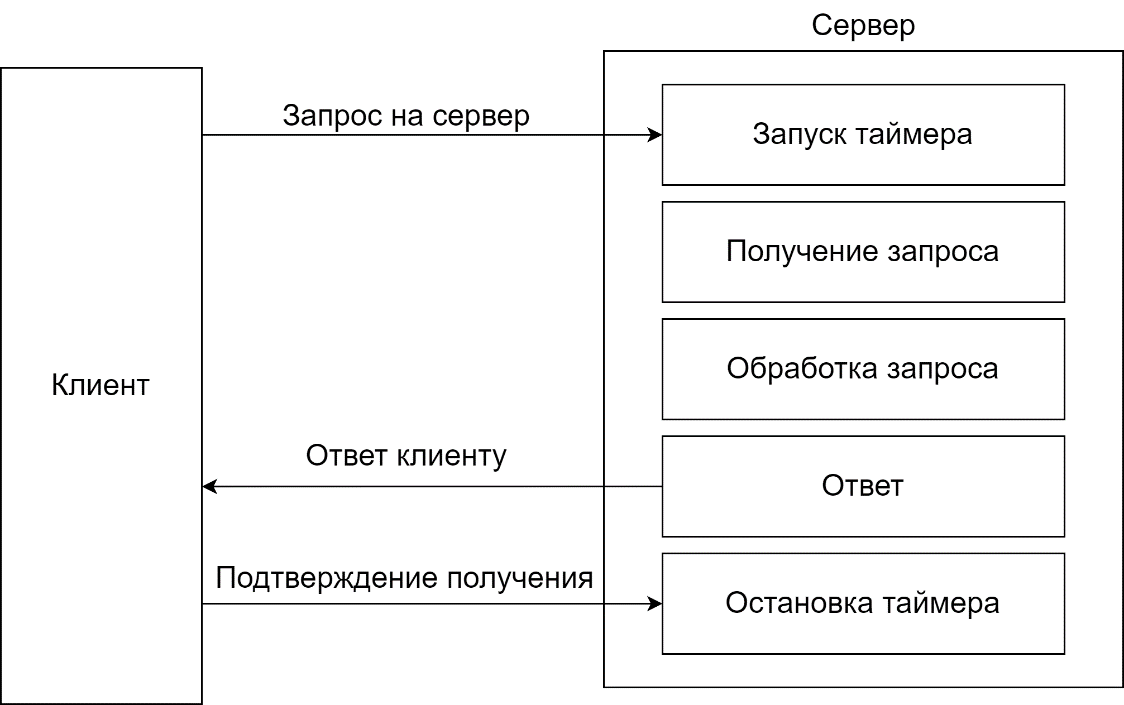


Рисунок 14 – Метрика с учетом времени доставки

Метрика второго типа нужна для того, чтобы мониторить время выполнения конкретных запросов на сервере.

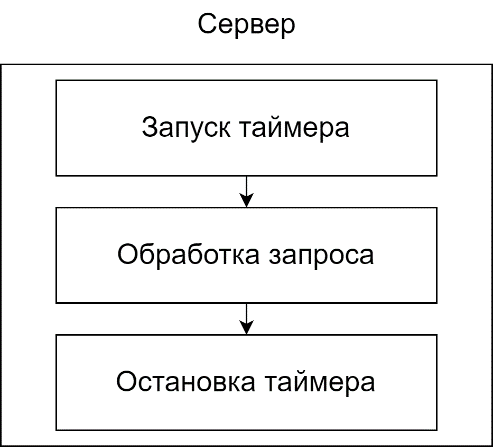


Рисунок 15 – Метрика без учета времени доставки

Если какой-то из запросов выполняется продолжительно время (больше 250 миллисекунд), то это является сигналом того, что метод нужно как-то ускорить (распараллелить, выделить ресурсы, сделать асинхронным).

* + 1. Алгоритм Синхронизации слов

Текущий алгоритм синхронизации на клиенте пользовательского словаря основан на исключении существующих слов.

Клиент отправляет уникальные идентификаторы пар слов конкретного пользователя, которых он хранит на сервер

Сервер приложения делает запрос в базу данных о имеющихся словах конкретного.

Новые слова, которые нужно добавить клиенту вычисляются исключением существующих на клиенте идентификаторов из общего множества идентификаторов.

Слова, которые необходимо удалить вычисляются путём исключения из идентификаторов клиента слов, которые существуют в базе данных.

Сервер отправляет клиенту списки слов которых недостаёт у клиента и список идентификаторов слов, которые нужно удалить.



Рисунок 16 – Диаграмма Вена

Таким образом если слова на клиенте — это множество а слова на сервере — это множество то слова **удаленные** это а **добавленные** это

На рисунке 16 представлена диаграмма Вена которая поясняет алгоритм представленный в виде последовательности действий.

Выводы

В главе 2 были рассмотрены инструменты проектирования серверной части инструментального комплекса, проведен сравнительный анализ одноранговой и клиент-серверной архитектур и выбрана для разработки серверной части инструментального комплекса клиент-серверная. Были спроектированы модули безопасности, индивидуального словаря пользователя и модуль общих наборов слов. В параграфе 2.4 были представлены разработанные алгоритмы подбора слов, оценки уровня игры, метрики, синхронизации слов и приведено их математическое обоснование.

1. Технологический раздел
   1. Обзор технологий используемых сервером ИК
      1. Методы и инструменты балансировки нагрузки

NGINX – это программное обеспечение с открытым исходным кодом для веб-обслуживания, обратного проксирования, кэширования, балансировки нагрузки, потоковой передачи мультимедиа и многого другого. Он начинался как веб-сервер, предназначенный для максимальной производительности и стабильности. В дополнение к своим возможностям HTTP-сервера, NGINX может также функционировать в качестве прокси-сервера для электронной почты (IMAP, POP3 и SMTP) и обратного прокси-сервера и балансировщика нагрузки для серверов HTTP, TCP и UDP. [6] Балансировка нагрузки – это способ масштабирования приложения и повышения его производительности и избыточности. NGINX, веб-сервер, который также может быть настроен как мощный балансировщик нагрузки.

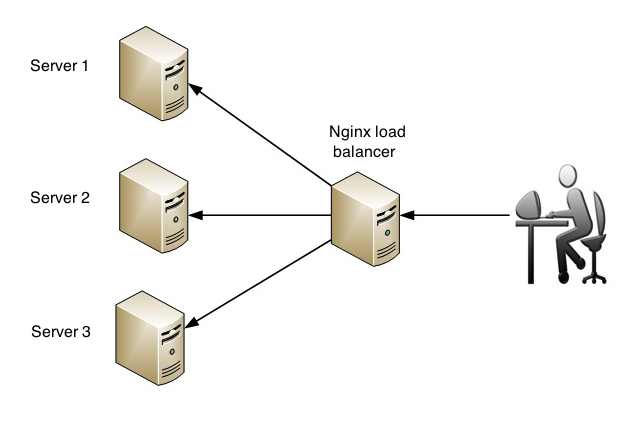


Рисунок 17 – Балансировка нагрузки NGINX

Типовая настройка для балансировки нагрузки выглядит следующим образом

http {  
 upstream myapp1 {  
 server srv1.example.com;  
 server srv2.example.com;  
 server srv3.example.com;  
 } server {  
 listen 80;  
 location / {  
 proxy\_pass http://myapp1;  
 }  
 }  
}

в этом примере имеется 3 экземпляра одного и того же приложения, выполняющего onsrv1, srv2 и srv3. Если метод балансировки нагрузки специально не настроен, то по умолчанию используется карусель (циклический перебор).

Так-же существуют другие механизмы балансировки нагрузки: по количеству подключений, балансировщик нагрузки сравнивает текущее количество активных подключений каждого сервера и отправляет запрос на сервер, имеющий наименьшее количество подключений. Это позволяет более справедливо контролировать нагрузку на экземпляры приложений в ситуации, когда выполнение некоторых запросов занимает больше времени.

Третий метод – хэш IP, следующий запрос от конкретного клиента всегда будет перенаправляться на один и тот же сервер, за исключением тех случаев, когда этот сервер недоступен. Был выбран именно этот метод, потому что при выполнении схожих запросов сервер кеширует ответ для конкретного пользователя что позволяет ускорить время ответа.

Четвертый метод – весовой баланс, каждому серверу можно задать вес который будет учитываться при балансировки Веса серверов не настраиваются в механизмах рассмотренных ранее, это означает, что четвертый метод независимый и не может быть скомбинирован с каким либо другим.

Наиболее целесообразно для серверной части инструментального комплекса использовать метод балансировки по хэшу IP.

* + 1. Методы сбора и отображения метрики

Prometheus – это инструментарий мониторинга и оповещения систем с открытым исходным кодом, первоначально созданный для SoundCloud. С момента своего создания в 2012 году многие компании и организации приняли Prometheus. Проект имеет активное сообщество разработчиков и пользователей. Теперь это автономный проект с открытым исходным кодом и поддерживается независимо от любой компании. Prometheus хорошо подходит для записи числовых временных рядов. Он подходит как для машинно-ориентированного мониторинга, так и для мониторинга высокодинамичных сервис-ориентированных архитектур. Для микросервисов его поддержка многомерного сбора данных и запросов является особой силой.

Prometheus предназначен для быстрого диагностирования проблем. Каждый сервер является автономным, не зависящим от сетевого хранилища или других удаленных служб.

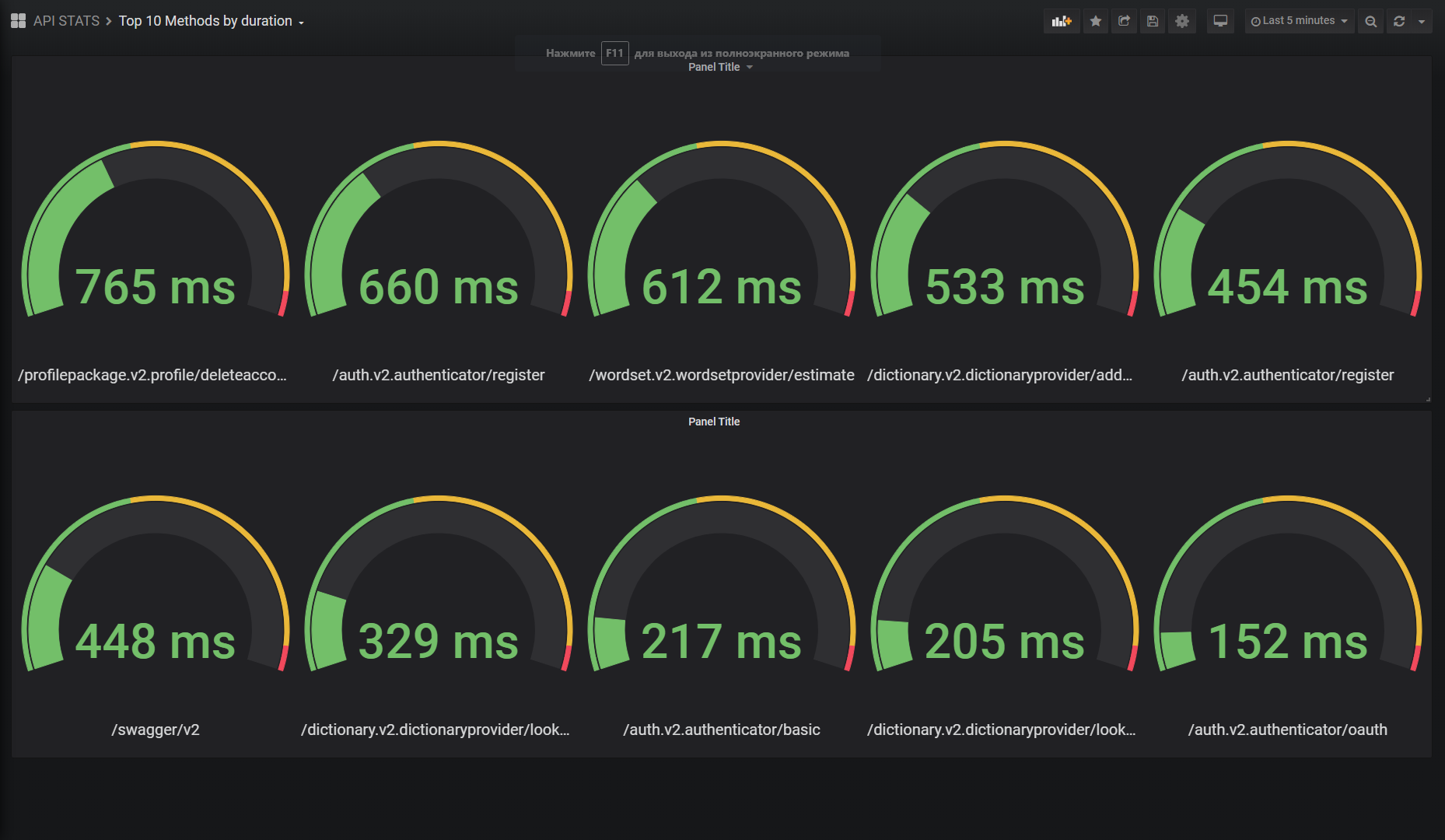


Рисунок 18 – Grafana дашборд времени ответа сервера

После сбора метрики с помощью сервера Prometheus её необходимо отобразить на наглядном дашборде, для этого используется Grafana. Она позволяет запрашивать, визуализировать, предупреждать и исследовать метрики независимо от того, где они хранятся. Другими сломами Grafana предоставляет централизированный, наглядный доступ ко всем источникам данных мониторинга.

* 1. Обзор инструментов разработки
     1. Azure DevOps

Azure DevOps платформа с бесплатным доступом нацеленная на непрерывную доставку ценностей конечным пользователям. В бесплатном пакете доступны все функции. Единственным ограничением является количество одновременно активных агентов. Агент – это машина, компьютер, который выполняет работу, заданную в системе. Используя Azure DevOps, можно повысить точность планирования, продуктивность сотрудничества и скорость поставки решений с помощью набора современных служб разработки.

CI/CD для любой платформы: создание, тестирование и развертывание решения на любом языке, в любом облаке или локальной среде. Параллельно выполнение задачи в Linux, macOS и Windows.

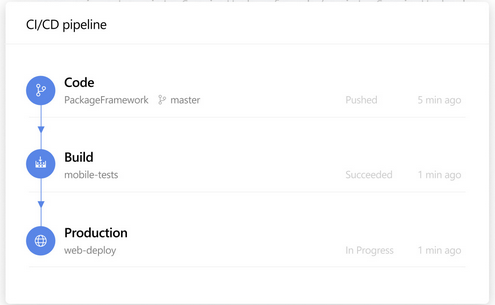


Рисунок 19 – Azure pipelines

Azure DevOps позволяет автоматизировать абсолютно весь процесс начинающийся с момента публикации кода в систему контроля версий. Все следующие шаги включающие сборку, тестирование, публикацию на удаленный сервер, публикацию информации о релизе и удаление старых релизов. Настраиваемый конвейер содержит готовые скрипты такие как: скачивание ветки из системы контроля версий git, архивация, разархивирование файлов, скачивание SDK необходимой версии, подписывание сертификатов и многое другое.

* + 1. Выбор СУБД

Компании выбирают системы с открытым исходным кодам, для того чтобы сберечь средства, но почти все крупные фирмы избирают платные СУБД для получения возможности современных методов в одном ряду с новыми методами шифрования и защиты информации.

**Стоимость**

Несмотря на то что существует бесплатная пробная версия реляционной СУБД MS SQL, для получения полного функционала и снятия ограничений необходимо приобрести лицензии для каждого запущенного экземпляра базы данных. MySQL же распространяется под бесплатной лицензией, это означает что весь доступный функционал доступен без каких-либо материальных затрат.

**Поддержка сообщества**. Огромное количество пользователей MySQL создает обширную бесплатную базу знаний в сети интернет, таким образом вместо того чтобы обращаться в официальную, платную техническую поддержку, зачастую большинство вопросов легко находятся на форумах сообщества.

Таблица 1 – Храктеристики таблиц

|  |  |  |
| --- | --- | --- |
| Характеристика | MySQL | Microsoft SQL Server |
| Расширяемость | Позволяет масштабироваться по мере роста | |
| Таблицы | стандартная реляционной табличная модель | |
| Синтаксис | Незначительные различия | |
| OS | Linux/Windows | |
| Популярность | Microsoft SQL Server и MySQL самые популярные субд | |
| Совместимость | В оригинале для PHP | В оригинале для .NET |
| Конфигурируемость | MyISAM and InnoDB | – |
| Цена | Бесплатная | Платная |
| LINQ | Разработки 3й стороны | Пакеты разработчика |
| Инструменты разработки | Enterprise Manager | Management Studio |

**Основные различий между MySQL и MS SQL Server**

Несмотря на то, что корпорация Microsoft с недавного времени обеспечивает запуск СУБД на всех трех популярных операционных системах. Некоторые функции всё ещё доступны только под ОС Windows. В противоположность этому, MySQL одинаково стабильно работает на Linux, Windows и Mac.

**Поддерживаемые Языки Программирования**. Обе СУБД поддерживают востребованные языки программирования: Java, C++, C#, Python, PHP, Ruby, VB, Deplhi, Go, R. Однако MySQL может в дополнении обеспечивает поддержку языков Scheme, Haskel, Perk, Eiffel. Очевидно что из-за большего спектра поддерживаемых языков MySQL собирает больше сообщество различных разработчиков.

**Подсистема хранилища.** В отличии от MS SQL, MySQL поддерживает подключаемые модули хранения данных. Таким образом не обязательно полностью обновлять СУБД чтобы получить новые функции хранения данных. Для получения нового, лучшего, функционала хранения данных в MS SQL придется обновлять полностью на новую версию СУБД.

**Фильтрация MySQL** позволяет осуществлять фильтрацию строк и таблиц но выборка из разных базы данных осуществляется по отдельности. SQL сервер обладает более гибкой технологии фильтрации через все базы данных, также сохраняя результаты в отдельной базе данных.

**Резервное копирование.**

SQL Server в отличии от MySQL не блокирует БД на время резервного копирования, что делает его более пригодным и устойчивым к резервному копированию и восстановлению, в то время как MySQL тратит огромные ресурсы на блокировку базы данных как на время копирования так и на время восстановления что делает процедуру более дорогой по времени и усилиям.

**Возможность остановить выполнение запроса.**

Сообщество отмечает главным недостатком MySQL сервера отсутствием возможности отменять запрос во время его выполнения. Необходимо остановить весь процесс, чтобы отменить выполнение запроса. MS SQL сервер лишен такого недостатка и легко может отменить запрос прямо в о время его выполнения. Транзакции – критически важная особенность MS SQL сервера делающая его значительна лучше конкурента.

**Безопасность**

Обе СУБД построены как двоичные коллекции. Пользователи MySQL во время работы взаимодействуют с файлами базы данных через бинарные файлы. Что позволяет им влиять на выполнение других процессов. SQL Server не дает другим процессам доступ к бинарным файлам напрямую. Пользователи должны выполнять определенные функции запустив экземпляр базы данных, что делает СУБД наиболее безопасным решением.

**Выпуски.**

У MySQL два основных выпуска «Community Server» и «Enterprise server» первый выпускается как открытое программное обеспечение второй с рядом закрытых, платных функций. MS SQL Server имеет несколько изданий: стандатрый, веб, рабочая группа, экспресс. Существуют и другие выпуски СУБД «Azure», «Compact», «developer », « embedded », « evaluation », «, fast track » «localDB»

**Как компонент программного стека**

Компании совершают выбор из выпусков MS SQL обращая внимание на количество одновременных подключений и других требований проекта. Также выбор зависит от предпочтений компаний проприоританых технологии в отличии от открытых. MySQL выбирают компании зачастую использующее классический стек LAMP (OS LINUX, Apache, PHP)

Схожесть двух полноценных СУБД заставляет разработчиков совершить нелегкий выбор в соответствии с требованиями проектов разработки,

Похожесть баз данных делает акцент на выбор по среде размещения и возможным материальным расходам. На Linux машинах хостинга по зачастую умолчанию установлен MySQL поскольку СУБД поставляется бесплатно и можно запускать столько баз данных сколько необходимо.

В целом SQL Server является дорогостоящим решением, с учетом платной лицензии и предоставлении одной базы на каждую лицензию.

Большинство разработчиков Windows работают с MSSQL, а разработчики Linux с MySQL. С учетом экономии денежных ресурсов и использования среды linux была выбрана СУБД mysql.

* + 1. Visual studio

Visual Studio 2019 – Классический инструмент разработки .NET, содержит новые функции, направленные на оптимизацию производительности разработчиков и совместной работы в команде. Упрощенная модель создания проектов путем клонирования из репозитория, а также инструменты повышения и качества кода. Функция поиска, ранее известная как быстрый запуск, была полностью переписана, чтобы сделать ее более быстрой и точной. Это обеспечивает возможность поиска с орфографическими ошибками.

Обновленный IntelliCode - это набор вспомогательных возможностей искусственного интеллекта, которые повышают производительность разработчиков: как контекстное IntelliSense, форматирование кода и вывод правил стиля.

CodeLens был функцией, предназначенной для Visual Studio Enterprise теперь так же доступен в Visual Studio 2019 Community. CodeLens показывает количество ссылок на тип или метод и информацию о модульных тестах, охватывающих метод. CodeLens позволяет легко найти ключевую информацию о типах.

Релиз 2019 года также включает в себя расширенные функции очистки кода проекта, которые позволяют не заботится о лишних подключенных пространств имен или стиле кода. Код автоматически может быть отформатирован путем использования этой функции.

* + 1. GRPC

Неэффективность протокола HTTP/1.1 – несжатые заголовки, отсутствие полноценной двусторонней связи, неэффективный подход к использованию ресурсов ОС, лишний трафик, лишние задержки и некоторые сложности описания модели данных REST с большим количеством неоднозначных вариантов возвращения нужных кодов [9]. gPRC призван решить эти проблемы используя стек технологий включающих в себя Protobuf, HTTP/2, статические пути. Protobuf является инструментом описания моделей данных, протобаф существовал за долго до gRPC. HTTP/2 – транспортный протокол, включает в себя методы сжатия, контроля трафика, инициаций событий с сервера, пере использование одного сокета для нескольких параллельный запросов. Поддержка популярных языков программирования и неограниченная возможность реализовать свою версию для любого языка.

Почему стоит использовать gRPC и gRPC-Web?

С помощью gRPC легко определить свою службу один раз .proto-файле и автоматически реализовать клиентские приложения на любом из поддерживаемых gRPC языков, которые. Всю сложность коммуникации между различными языками и средами решается за счет gRPC. Также получаются все преимущества работы с буферами протоколов, включая эффективную сериализацию, обновление интерфейса. gRPC-Web позволяет получить доступ к службам gRPC, построенным таким образом, из браузеров, использующих идиоматический API.

Таким образом, для уменьшения передаваемого трафика и, одновременно, ускорения процесса разработки будет использовано решение от компании Google – gRPC. gRPC – это высокопроизводительный RPC-фреймворк с открытым исходным. Создан для эффективного подключения службы в центрах обработки данных и между ними. Поддерживает балансировку нагрузки, трассировку, проверку работоспособности и аутентификацию. Так-же отлично подходит в сетях с низким качеством связи, для подключения устройств интернета вещей, мобильных приложений и браузеров к серверным службам API.

* + 1. .Net

.NET Core – модульный, кроссплатформенная платформа с открытым сходным кодом, огромный шаг компании Microsoft в сторону сообщества. Оптимизированная для разработки производительных веб-приложений теперь может быть запущена на любых устройствах от мобильных устройств на ARM процессорах, до огромных дата-центров.

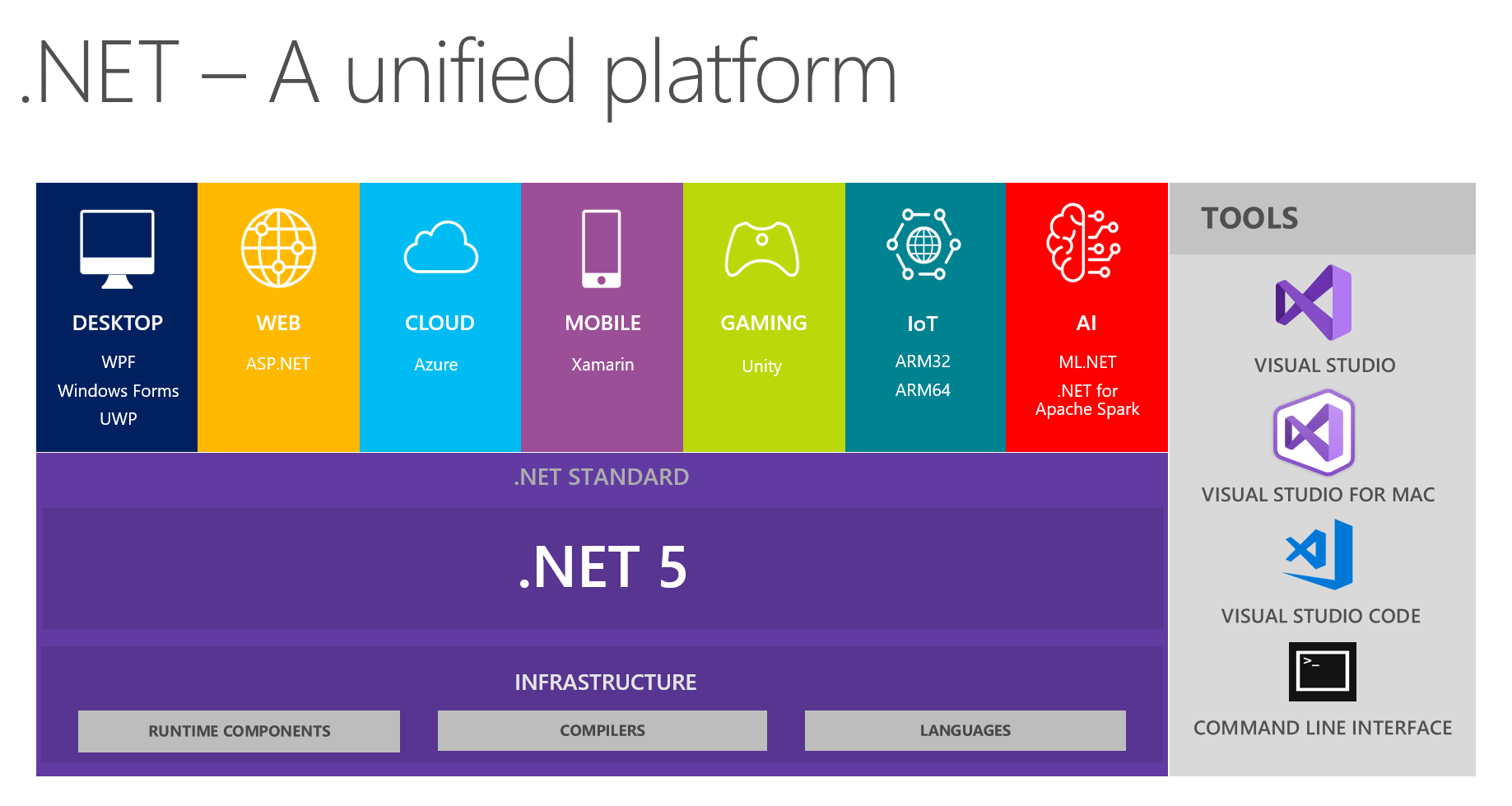


Рисунок 20 – Архитектура .NET 5

Следующий крупный релиз – .NET 5 объединение .NET в единую платформу, включая приложения для мобильных устройств, разработку веб приложений, настольных компьютеров, приложений машинного обучения и интернета вещей. Впервые вся платформа будет использовать унифицированный BCL (библиотеки базовых классов) для всех моделей приложений. Также с учетом наличия средств разработки облачных приложений и игр языки платформы .NET будут использовать единую библиотеку базовых классов для всех сфер разработки.

* + 1. xUnit

Фреймворк для тестирования, мало чем отличается от аналогичных за исключением соглашением в коде которые более соответствуют стандартам тестирования. Для создания тестовых классов по правилам сообщества следует создать папку Tests и в ней разместить проект тестирования и таким же именем как и тестируемый проект но с дополнительным окончанием «.Tests». Для выделения тестовых методов используются атрибуты «Fact» и «Theory» для метода без входных значений и для метода с входными тестируемыми значениями соответственно. Фреймворк был выбран из-за его поддержки CI/CD.

* + 1. Serilog

Serilog обеспечивает базовое ведение журнала диагностики для разных способов вывода информации: консоль, файлы, Amazon, Azure и т. д. Кроме первоначальной конфигурации потребителей вывода сообщений для логирования определяются ещё два пункта – уровень логирования и текст сообщения. Существуют такие уровни как информационное сообщение, предупреждение, ошибка, критическая ошибка. Основное удобство использование этого инструмента разработки заключается в том, что можно не тратить время на разработку своей системы логирования, а использовать уже готовый, бесплатный, хорошо протестированный, асинхронных инструмент логирования. Также наличие возможности легко конфигурировать потребителей вывода логов делает возможным записывать журналы сразу в несколько мест.

Таким образом выбранные инструменты (Azure DevOps, MySQL, Visual studio, gRPC, .Net, xUnit, Serilog) обладают следующими свойствами: открытая доступ, понятная документация, каждый инструмент обеспечивается долгосрочной поддержкой, поддерживают эффективную реализацию.

* 1. Описание используемых паттернов

При реализации серверной системы основным паттерном для открытых методов API был выбран паттерн проектирования Посредник (англ. Mediator) –обеспечивающий взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга [11]. Суть паттерна посредник состоит в том, чтобы "определить объект, который инкапсулирует то, как взаимодействует набор объектов". Он способствует свободному сцеплению, удерживая объекты от явного обращения друг к другу, и позволяет их взаимодействие варьироваться независимо.[12] клиентские классы могут использовать медиатор для отправки сообщений другим клиентам, а также могут получать сообщения от других клиентов через событие в классе медиатора.

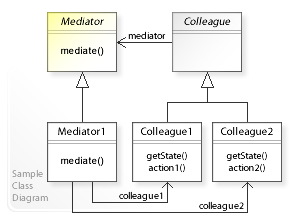


Рисунок 21 – UML диаграмма паттерна посредник

Тесно связанные объекты трудно реализовать, изменить, протестировать и повторно использовать, потому что они ссылаются на множество различных объектов. Паттерн посредник выбран для решения такие этой проблемы. А именно для возможности независимо изменять взаимодействие между множеством объектов и для избегания тесной связи между множеством взаимодействующих объектов.

Необходимость ввести этот паттерн может возникнуть, когда в коде оказывается слишком много прямых связей с другими частями. Посредник, в таком случае, способствует свободной связи между объектами, гарантируя, что вместо того, чтобы они явно ссылались на другие объекты, все их взаимодействия будут координироваться через центральную точку, которая будет посредником. Это помогает улучшить удобство использования компонентов.

Самое большое преимущество, которое обеспечивает паттерн посредник это тестопригодность классов и методов.

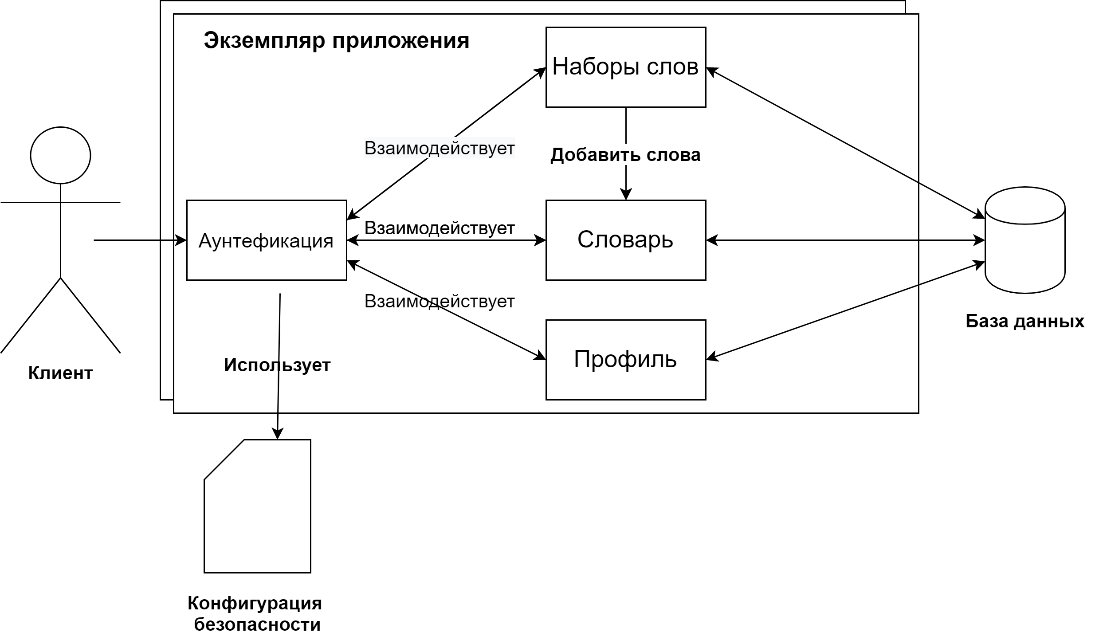


Рисунок 22 – Взаимодействие внутри системы

На рисунке 21 представлена часть основных компонентов системы, с которыми непосредственно взаимодействует клиент.

* 1. Описание реализации базы данные

В параграфе 2.3 была спроектирована схема в виде диаграммы использования (рисунок 9 и 10). С помощью ORM Entity Framework была создана база данных представленная на рисунке 23.

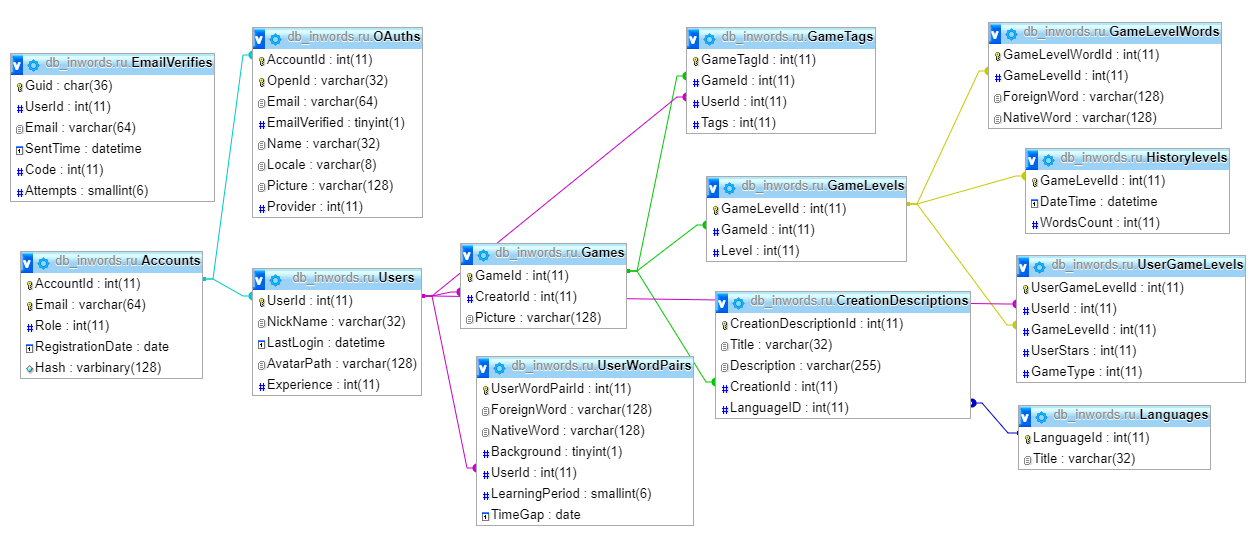


Рисунок 23 – Схема базы данных

База данных состоит из 13 сущностей 12 из которых непосредственно необходимы для работы сервера и одна для подтверждения пользовательского почтового ящика. В качестве СУБД используется MariaDB

Выводы

В главе 3 были выбраны технологии и инструменты реализации серверной части инструментального комплекса. Представлен обзор инструментов Azure DevOps, MySQL, Visual Studio, gRPC, .NET, xUnit, Serilog. Проведен сравнительный анализ СУБД MS SQL и MySQL и выбрана для реализации серверной части инструментального комплекса MySQL. В параграфе 3.3 были представлены паттерны, используемые в реализации серверной части инструментального комплекса и в параграфе 3.4 создана базы данных, на основе библиотеки Swagger был реализован пользовательский интерфейс.

1. Тестирование
   1. Описание тестирования

Считается, что ПП обладает высоким качеством, если его характеристики соответствуют тем требованиям, которые ему предъявляются со стороны заказчика, пользователя и стандартов.

1) ПП должен работать в соответствии с функциями покрытия контента, указанными в описании ПП и пользовательской документации.

2) Работа ПП должна соответствовать функциям надежности, указанным в пользовательской документации.

3) Работа ПП должна соответствовать характеристикам безопасности, указанным в пользовательской документации.

4) Успешная установка и правильная работа ПП должны подтверждаться для всех поддерживаемых платформ и систем, указанных в описании ПП.

5) ПП должен использовать те конфигурации программно-аппаратных средств и те ресурсы (например, объем оперативной памяти, объем жесткого диска, тип процессора) которые указаны в описании ПП и пользовательской документации.

6) Должно быть предусмотрено масштабирование затрат памяти и вычислительных ресурсов ПП.

7) ПП должен предоставлять средства для оценки влияния ПП на заданные цели соответствия.

8) ПП должен работать в соответствии с функциями покрытия контента, указанными в описании ПП и пользовательской документации.

9) Если ПП используются параметры, ограничивающие полное функциональное покрытие, пользователю должно быть известно текущее покрытие контента.

Для тестирования используется Azure DevOps Pipelines, эта система делает автоматическое восстановление, сборку, и тестирование программного продукта перед каждым выпуском, или в определенный момент времени, например при внесении изменений в рабочую ветку системы контроля версий (GIT).

Текущий уровень покрытия тестами программы 32.27% И при этом проходят все тесты.

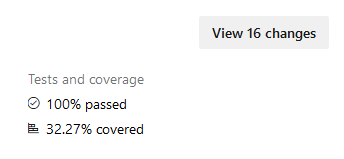


Рисунок 24 – Покрытие тестами

Если один из тестов не проходит, то система показывает это, и указывает место возникновения ошибки.

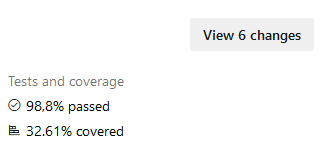


Рисунок 25 – Поктытие тестами, провален 1 тест.

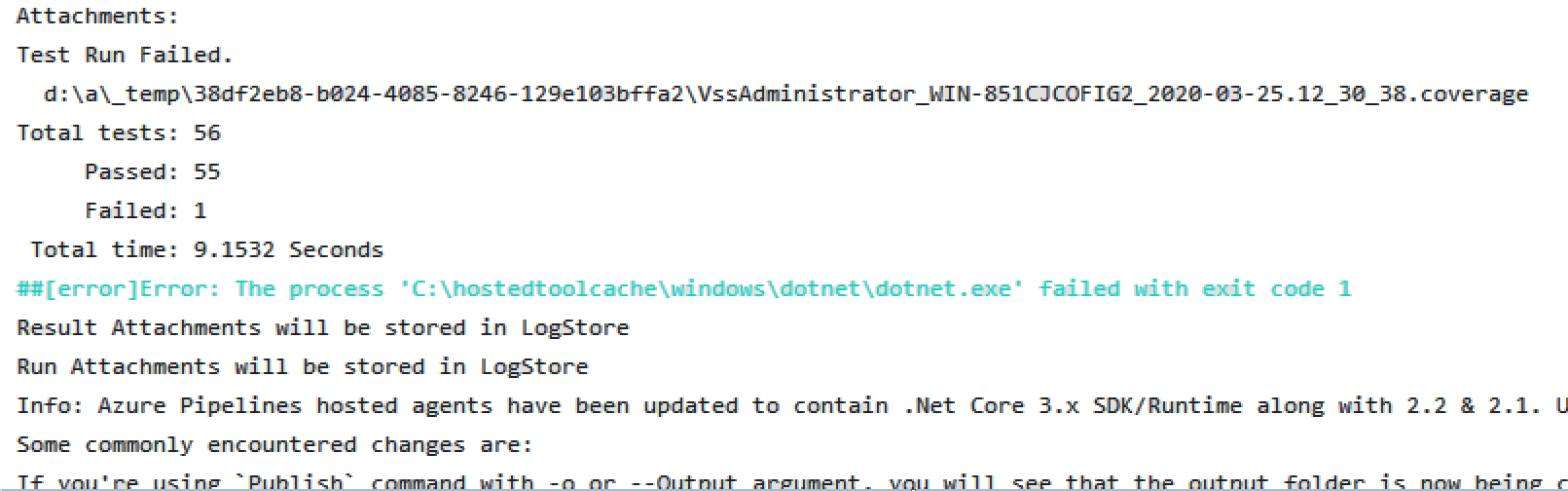


Рисунок 26 – В последнем блоке тестирования провален 1

После исправления программы, можно убедиться, что все тесты проходят.

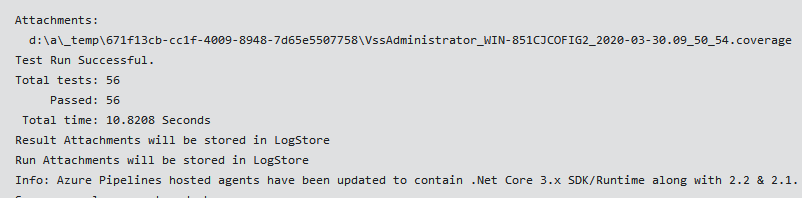


Рисунок 27 – Блок протестирован успешно

С помощью Azure DevOps можно анализировать тестовые случаи за последний месяц и выявить в каких ситуация проваливаются тесты чаще всего.

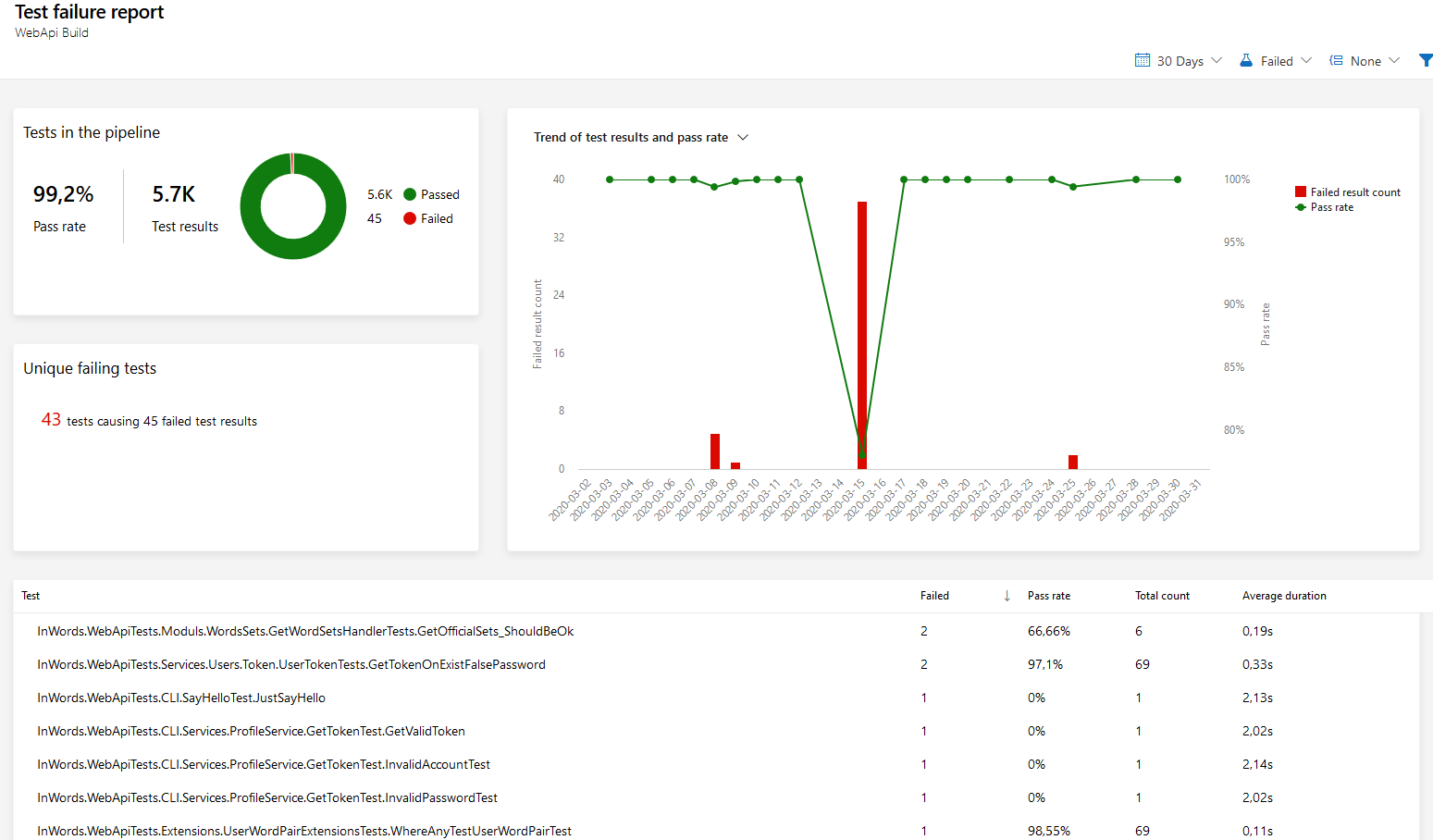


Рисунок 28 – Отчёт о тестировании за месяц

Из отчёта видно, что при внедрении нового функционала в один из дней, провалилось много тестов, это значит, что большая часть функционала перестала работать, но эта сборка не попала на рабочий сервер благодаря покрытию тестами. Вскоре все ошибки были исправлены.

Хорошее покрытие тестами серверной части инструментального комплекса обеспечило стабильность и качество работы сервера.

* 1. Описание непрерывной интеграции

Все этапы автоматизации непрерывной интеграции описываются на языке YAML. Сразу после того как код попадает в систему контроля версий, на сервисе Azure DevOps запускается чистый контейнер который производите следующие инструкции (Код представлен в приложении 1) Для сборки серверного приложения .NET Core выполняются следующие этапы

1. Установка версии .NET Core в контейнер которая установлена на сервере, это позволит отсеять те случаи, когда после сборки в системе непрерывной интеграции приложение не работает на сервере из-за того, что вышла новая версия фреймворка.
2. Восстановление зависимостей сторонних библиотек
3. Сборка решение
4. Тестирование
5. Публикация собранных артефактов
6. Публикация результатов тестирования

Публикация результатов тестирования необходима для отслеживания покрытия тестирования (рисунок 29).

|  |  |
| --- | --- |
|  |  |

Рисунок 29 – Отчет о покрытии

Отчет условно выделяет строки зеленым цветом если они были выполнены во время выполнения модульных тестов и красным цветом выделены те строки, которые возможно покрыть тестами, но на данный момент не покрыты.

Кроме конвейера сборки, существует так же конвейер публикации, который собирается из готовых модулей сервиса и состоит из нескольких этапов (рисунок 30).

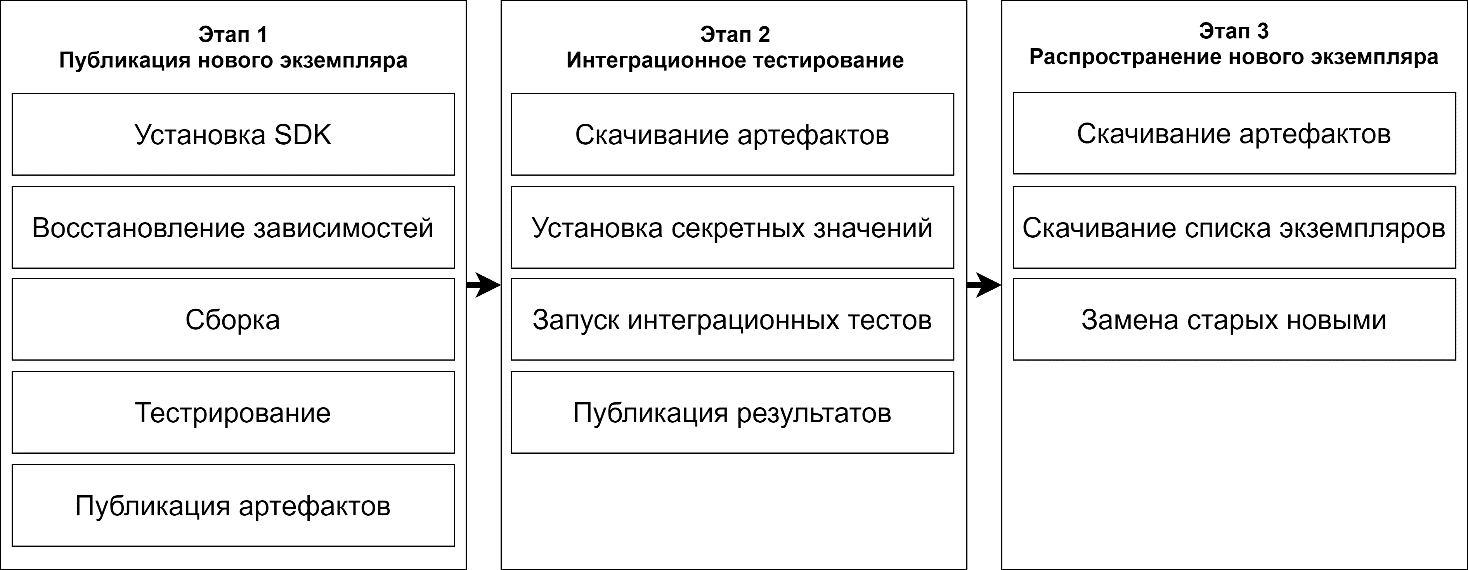


Рисунок 30 – Конвейер выпуска

Первый этап — это публикация артефактов сборочного конвейера на рабочий сервер через FTP. Во время этого этапа происходит несколько шагов

1. Артефакты разархивируются
2. В корень сборки копируется файл с паролями подключения из секретного хранилища
3. Сборка архивируется
4. Временно выключается старая версия приложения
5. Загружаются файлы на сервер
6. Включается новая версия

На этапе два – новая версия приложения проходит интеграционное тестирование и если оно прошло успешно, то на этапе три копия этого приложения распространяется на все экземпляры приложения запущенные на сервере.

Выводы

По результатам четвертой главы были проведено модульное и интеграционное тестирование, разработан конвейер непрерывной интеграции с автоматическим тестированием.

1. Экономический раздел
   1. Организация и планирование работ по теме

В составе работы задействовано 3 человека:

1. руководитель ВКР – отвечает за грамотную постановку задачи, контролирует отдельные этапы работы, вносит необходимые коррективы и оценивает выполненную работу в целом;
2. консультант по специальной части ВКР – отвечает за консультирование в области технической части проекта: предлагает возможные пути решения задач, выбора инструментов разработки, способов разработки;
3. разработчик — реализация всех поставленных задач, в том числе проведение тестирования готового продукта и подготовка проектной документации.

Состав задействованных в работе участников представлен на схеме.

Консультант

Разработчик

Руководитель

* + 1. Организация работ

На разработку отводится 95 рабочих дней.

Этапы разработки представлены в табл. Таблица 2.

Таблица 2. Этапы разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название этапа | Исполнитель | Трудоемкость,  чел/дни | Продолжительность работ, дни |
| 1 | Разработка и утверждение технического задания | Руководитель | 5 | 5 |
| 2 | Технические предложения | Руководитель | 7 | 7 |
| Консультант | 3 |
| 3 | Эскизный проект: |  |  | 21 |
| 3.1 | Анализ исходных данных и требований | Разработчик | 9 |
| 3.2 | Постановка задачи | Консультант | 5 |
| 3.3 | Разработка общего описания алгоритма функционирования | Руководитель | 2 |
| Разработчик | 7 |
| 4 | Технический проект: |  |  | 15 |
| 4.1 | Определение формы представления входных и выходных данных | Руководитель | 2 |
| Разработчик | 5 |
| 4.2 | Разработка структуры программы и логической структуры базы данных | Руководитель | 2 |
| Консультант | 2 |
| Разработчик | 10 |
| 5 | Рабочий проект: |  |  | 47 |
| 5.1 | Программирование и отладка программы | Разработчик | 24 |
| 5.2 | Испытание программы | Разработчик | 4 |
| 5.3 | Корректировка программы по результатам испытаний | Разработчик | 5 |
| 5.4 | Подготовка технической документации на программный продукт | Консультант | 3 |
| Разработчик | 7 |
| 5.5 | Сдача готового продукта и внедрение | Руководитель | 2 |
| Консультант | 2 |
| Разработчик | 7 |
| Итого | | | 113 | 95 |

* + 1. График проведения работ

Календарный график исполнения работы представлен на рисунке Рисунок 31. Из рисунка Рисунок 31 так же видно, что общий срок разработки составит 95 дней.

Рисунок 31 – Календарный график исполнения работы

* 1. Расчёт стоимости проведения работ

1 статья «Материалы, покупные изделия и полуфабрикаты + ТЗР (20%) от ∑ итого по материалам

себестоимость

2 статья «Специальное оборудование» - затрат нет

3 статья «Основная заработная плата»

4 статья «Дополнительная заработная плата» 20% от основной заработной платы

5 статья «Страховые отчисления» - 30% от ФОТ

6 статья «Командировочные расходы» - затрат нет

7 статья «Контрагентские услуги» - затрат нет

8 статья «Накладные расходы» - 200% от основной заработной платы

9 статья «Прочие расходы»

В выпускной квалификационной работе объем затрат на НИР и ОКР был проведен методом калькулирования.

* + 1. 1 статья «Материалы, покупные изделия и полуфабрикаты»

Расчет затрат на материалы, покупные изделия и полуфабрикаты приведен в табл. Таблица 3.

Таблица 3. Затраты на материалы, покупные изделия и полуфабрикаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № пп | **Наименование материалов** | Единицы измерения | Количество | Цена за единицу (руб) | Стоимость (руб) |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Флешка 16 Гб | шт | 1 | 360 | 360 |
| 2 | Бумага А4 | пачка | 1 | 225 | 225 |
| 3 | Картридж для принтера | шт | 1 | 1240 | 1240 |
| 4 | Ручка | шт | 10 | 17 | 170 |
| 5 | Карандаш | шт | 10 | 7 | 70 |
| Итого материалов | | | | | 2 065‬ |
| Транспортно-заготовительные расходы | | | | | 413 |
| Итого | | | | | 2478 |

* + 1. 2 статья «Специальное оборудование»

Расходы на специальное оборудование отсутствуют.

* + 1. 3 статья «Основная заработная плата»

Расчет основной заработанной платы приведен в табл. Таблица 4.

Таблица 4. Основная заработная плата

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № пп | Наименование этапа | Исполнитель (должность) | Мес. оклад (руб) | Трудоемкость (чел/дни) | Оплата за день (руб) | Оплата за этап (руб) |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ТЗ | Руководитель | 40 000 | 5 | 1818 | 9090 |
| 2 | ТП | Руководитель | 40 000 | 7 | 1818 | 12726 |
| Консультант | 50 000 | 3 | 2272 | 6816 |
| 3 | Эскизный проект | Руководитель | 40 000 | 2 | 1818 | 3636 |
| Консультант | 50 000 | 5 | 2272 | 11360 |
| Разработчик | 25 000 | 16 | 1136 | 18176 |
| 4 | Технический проект | Руководитель | 40 000 | 4 | 1818 | 7272 |
| Консультант | 50 000 | 2 | 2272 | 4544 |
| Разработчик | 25 000 | 15 | 1136 | 17040 |
| 5 | Рабочий проект | Руководитель | 40 000 | 2 | 1818 | 3636 |
| Консультант | 50 000 | 5 | 2272 | 11360 |
| Разработчик | 25 000 | 47 | 1136 | 53392 |
| Итого | | | | | | 159 048 |

Оплата за день рассчитывается делением месячного оклада на 22 дня.

* + 1. 4 статья «Дополнительная заработная плата»

На эту статью относятся выплаты, предусмотренные законодательством о труде за неотработанное по уважительным причинам время; оплата очередных и дополнительных отпусков; времени, связанного с выполнением государственных и общественных обязанностей; выплата вознаграждения за выслугу лет и т.п. (в среднем она составляет 20-30% от суммы основной заработной платы).

ДЗП = 159 048 × 0,2 = 31 809 руб.

Дополнительная заработная плата научного и производственного персонала составляет по проекту 31 809 руб.

* + 1. 5 статья «Страховые отчисления»

Отчисления на социальные нужды составляют 30% от фонда оплаты труда (ФОТ), который состоит из основной и дополнительной заработной платы.

ФОТ = ОЗП + ДЗП = 159 048 + 31 809 = 190 857 руб.

СВ = ФОТ × 30% = 190 857 × 0,30 = 57 258 руб.

* + 1. 6 статья «Командировочные расходы»

Расходы по данному разделу отсутствуют.

* + 1. 7 статья «Контрагентские услуги»

В процессе разработки данного проекта услуги сторонних организаций не использовались.

* + 1. 8 статья «Накладные расходы»

Накладные расходы – это затраты предприятия, не связанные напрямую с выполнением задач проекта. Они не входят в оплату труда и материалов. Обычно определяются как 200-300% от показателя основной заработной платы.

НР = ОЗП х 200% = 159 048 × 2,0 = 318 096 руб.

* + 1. 9 статья «Прочие расходы»

Расчет прочих расходов приведен в табл. Таблица 5.

Таблица 5. Прочие расходы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № пп | Наименование расходов | Единицы измерения | Количество | Цена за единицу (руб) | Стоимость (руб) |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Аренда домена | год | 1 | 900 | 900 |
| 2 | Аренда выделенного сервера | год | 1 | 3600 | 3600 |
| Итого | | | | | 4500 |

* + 1. Полная себестоимость проекта

Расчет полной себестоимости проекта приведен в табл. Таблица 6.

Таблица 6. Полная себестоимость проекта

|  |  |  |
| --- | --- | --- |
| № пп | Номенклатура статей расходов | Затраты (руб) |
| 1 | 2 | 3 |
| 1 | Материалы, покупные изделия и полуфабрикаты (за вычетом отходов) | 2600 |
| 2 | Специальное оборудование для научных (экспериментальных) работ | - |
| 3 | Основная заработная плата научного и производственного персонала | 159 048 |
| 4 | Дополнительная заработная плата научного и производственного персонала | 31 809 |
| 5 | Страховые взносы в социальные фонды | 57 258 |
| 6 | Расходы на научные и производственные командировки | - |
| 7 | Оплата работ, выполненных сторонними организациями и предприятиями | - |
| 8 | Прочие прямые расходы | 4500 |
| 9 | Накладные расходы | 318 096 |
| Итого | | 573 311 |

Прибыль будет равна: П = 573 311 × 26% = 149 061 руб.

Рассчитаем налог на добавочную стоимость (НДС) в размере 20%:

НДС = (С + П) × 20% = (573 311 + 149 061) × 0,2 = 144 474 руб.

Таким образом, договорная цена будет представлять собой:

ДЦ = С + П + НДС = 573 311 + 149 061 + 144 474 = 866 846 руб.

# **Заключение**

Поставленная задача полностью выполнена в Работа была проделана в соответствии с техническим заданием.

В первой главе был проведен анализ систем обучения английскому языку, обладающих свойствами: открытое API, REST подключение, gRPC подключение, готовая база материала и сформулирована задача разработки.

В второй главе были рассмотрены инструменты проектирования серверной части инструментального комплекса, проведен сравнительный анализ одноранговой и клиент-серверной архитектур. Для разработки серверной части инструментального комплекса выбрана клиент-серверная архитектура. Были спроектированы модули безопасности, индивидуального словаря пользователя и модуль общих наборов слов. Были представлены разработанные алгоритмы подбора слов, оценки уровня игры, метрики, синхронизации слов и приведено их математическое обоснование.

В третьей главе были рассмотрены инструменты разработки серверной части инструментального комплекса и проведен сравнительный анализ СУБД MS SQL и MySQL и выбрана для разработки серверной части инструментального комплекса MySQL, представлены паттерны разработки и описана схема базы данных.

По результатам четвертой главы были проведено модульное и интеграционное тестирование, разработан конвейер непрерывной интеграции с автоматическим тестированием.

В пятой главе была обоснована экономическая целесообразность проекта.

В результате выполнения выпускной квалификационной работы был реализован сервер инструментального комплекса «Инструментальный комплекс создания программ для обучения английскому языку отдельных категорий пользователей. Серверная часть», реализовано взаимодействия отдельных модулей внутри системы, а также конвейер автоматического тестирования.

# **Список литературы**

1. Клиент-серверная архитектура // Технопедия URL: https://www.techopedia.com/definition/438/clientserver-architecture (дата обращения: 01.06.2020).
2. Закон Мура // Википедия URL: https://en.wikipedia.org/wiki/Moore%27s\_law (дата обращения: 01.06.2020).
3. Манифест гибкой разработки // Википедия URL: https://ru.wikipedia.org/wiki/Agile\_Manifesto (дата обращения: 01.06.2020).
4. Рыночная доля устройств // Statcounter URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet (дата обращения: 01.06.2020).
5. Операционная система публичных серверов в интернете // Википедия URL: https://en.wikipedia.org/wiki/Usage\_share\_of\_operating\_systems#Public\_servers\_on\_the\_Internet (дата обращения: 01.06.2020).
6. Nginx URL: https://www.nginx.com/resources/glossary/nginx/ (дата обращения: 01.06.2020).
7. Балансировка нагрузки // Медиум URL: https://medium.com/@tasnuva2606/http-load-balancing-using-nginx-869ca9377fb9 (дата обращения: 01.06.2020)
8. Что нового в Visual Studio 2019 // Майкрасофт URL: https://docs.microsoft.com/en-us/archive/msdn-magazine/2018/connect/visual-studio-what-s-new-in-visual-studio-2019 (дата обращения: 01.06.2020).
9. gRPC — фреймворк от Google для удалённого вызова процедур // Хабр URL: https://habr.com/ru/company/infopulse/blog/265805/ (дата обращения: 01.06.2020).
10. React and Google OAuth with .NET Core backend // Medium URL: https://medium.com/mickeysden/react-and-google-oauth-with-net-core-backend-4faaba25ead0 (дата обращения: 01.06.2020).
11. Соль (Криптография) // Википедия URL: https://en.wikipedia.org/wiki/Salt\_(cryptography) (дата обращения: 01.06.2020).
12. Кривая забывания // Википедия URL: https://en.wikipedia.org/wiki/Forgetting\_curve (дата обращения: 01.06.2020).
13. Ruger, Henry A. (1913). Memory: A Contribution to Experimental Psychology страница 22
14. Murre J.M.J., Dros J. (2015) "Replication and Analysis of Ebbinghaus’ Forgetting Curve". PLOS ONE 10(7): e0120644. doi: 10.1371/journal.pone.0120644
15. Rubin DC, Hinton S, Wenzel AE (1999) The precise time course of retention. Journal of Experimental Psychology: Learning, Memory, and Cognition 25: 1161–1176.
16. Mediator // Wikipedia URL: <https://en.wikipedia.org/wiki/Mediator_pattern> (дата обращения 18.05.2020)
17. Гавриленко Т.Ю., Григоренко О.В., Ткаченко Е.К. Корпоративная экономика и финансы. М.: РУСАЙНС, 2016-125с.
18. Григоренко О.В., Садовничая И.О., Мыльникова А. Экономика предприятия и управление организацией М.: РУСАЙНС, 2017-235с
19. Таненбаум Э., Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Ван Стеен; пер. с англ. – СПб.: Питер, 2003. – 876 с.
20. Распределенные системы. Паттерны проектирования /Бёрнс Б.. — Питер: Издательский дом Питер, 2019. — 224с.
21. Чистая архитектура. Искусство разработки программного обеспечения — СПб: Питер, 2018. — 352с.: ил. – (Серия «Библиотека программиста»).
22. Microsoft Visual C#. Подробное руководство /Джон шарп. — 8-е издание. — СПб: Питер, 2017. — 848с.
23. Внедрение зависимостей в .NET /Марк Симан. — СПб: Питер, 2018. — 464с.
24. Git для профессионального программиста. Подробное описание самой популярной системы контроля версий /Чакон Скотт, Штрауб Бен. — СПб: Питер, 2018. — 496с.
25. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов /Прайс М.. — 3-е издание. — СПб: Питер, 2019. — 640с.
26. Принципы, паттерны и методики гибкой разработки на языке C# /Мартин Роберт, Мартин Мика. — High Tech. — Москва: Символ-Плюс, 2011. — 768с.

Приложение 1 – Код непрерывной интеграции

# ASP.NET Core

# Build and test ASP.NET Core projects targeting .NET Core.

# Add steps that run tests, create a NuGet package, deploy, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

schedules:

- cron: "0 0 \* \* \*"

  displayName: Daily midnight Web API build

  branches:

    include:

    - develop

pr:

  branches:

    include:

    - webapi-release

    - develop

  paths:

    include:

    - WebApi/\*

trigger:

  branches:

    include:

    - webapi-release

    - develop

  paths:

    include:

    - WebApi/\*

variables:

    major: 2.0

    incremental: $[counter(variables['major'], 150)]

jobs:

- job: BuildAndTest

  timeoutInMinutes: 10

  pool:

    vmImage: 'windows-2019'

  variables:

    buildConfiguration: '-c Release'

    testConfiguration: '$(buildConfiguration) --collect "Code coverage" '

  steps:

  - task: NuGetCommand@2

    displayName: 'NuGet restore packages'

    inputs:

      command: 'restore'

      restoreSolution: '\*\*/InWords.WebApi/InWords.WebApi.sln'

      feedsToUse: 'select'

      vstsFeed: 'http://nuget.org/'

  - task: DotNetCoreCLI@2

    displayName: 'build any OS Release'

    inputs:

      command: build

      projects: '\*\*/InWords.WebApi/InWords.WebApi.csproj'

      arguments: '$(buildConfiguration)'

  # Run all tests with "/p:CollectCoverage=true /p:CoverletOutputFormat=cobertura" to generate the code coverage file

  - task: DotNetCoreCLI@2

    displayName: dotnet test

    inputs:

      command: test

      arguments: '$(BuildConfiguration) --collect:"XPlat Code Coverage" --settings D:\a\1\s\WebApi\InWords.WebApi\coverlet.runsettings'

      projects: 'D:\a\1\s\WebApi\InWords.WebApi\\*\*\\*Tests.csproj'

      nobuild: true

  # Generate the report using ReportGenerator (https://github.com/danielpalme/ReportGenerator)

  # First install the tool on the machine, then run it

  - script: |

      dotnet tool install -g dotnet-reportgenerator-globaltool

      reportgenerator -reports:D:\a\\*\*\coverage.cobertura.xml -tag:$(Build.BuildNumber) -targetdir:$(Build.SourcesDirectory)/CodeCoverage -reporttypes:HtmlInline\_AzurePipelines;Cobertura

    displayName: Create Code coverage report

  # Publish the code coverage result (summary and web site)

  # The summary allows to view the coverage percentage in the summary tab

  # The web site allows to view which lines are covered directly in Azure Pipeline

  - task: PublishCodeCoverageResults@1

    displayName: 'Publish code coverage'

    inputs:

      codeCoverageTool: Cobertura

      summaryFileLocation: '$(Build.SourcesDirectory)/CodeCoverage/Cobertura.xml'

      reportDirectory: '$(Build.SourcesDirectory)/CodeCoverage'

  # Publish artifacts stage

  - task: PublishPipelineArtifact@1

    displayName: 'Publish artifacts: \_WebApiTests'

    inputs:

      path: $(System.DefaultWorkingDirectory)/WebApi/

      artifact: \_WebApiTests

    condition: succeeded()

- job: DeployReadyToRun

  condition: and(succeeded(), eq(variables['build.sourceBranch'], 'refs/heads/webapi-release'))

  timeoutInMinutes: 10

  pool:

    vmImage: 'ubuntu-latest'

  steps:

  - task: UseDotNet@2

    displayName: 'Install .net core 3.1.300'

    inputs:

      packageType: sdk

      version: '3.1.300'

      installationPath: $(Agent.ToolsDirectory)/dotnet

  - task: DotNetCoreCLI@2

    inputs:

      command: 'publish'

      publishWebProjects: false

      projects: '\*\*/InWords.WebApi/InWords.WebApi.csproj'

      arguments: '-c Release -o $(Build.ArtifactStagingDirectory)/linux64 -p:PublishReadyToRun=true -p:PublishTrimmed=true -r linux-x64 --self-contained true'

      modifyOutputPath: false

  - task: PublishBuildArtifacts@1

    inputs:

      PathtoPublish: '$(Build.ArtifactStagingDirectory)/linux64'

      ArtifactName: '\_API-Linux64-Ready2Run'

      publishLocation: 'Container'

    condition: succeeded()

  - task: GitHubRelease@1

    inputs:

      gitHubConnection: 'Alword'

      repositoryName: 'inwords/InWords'

      action: 'create'

      target: '$(Build.SourceVersion)'

      tagSource: 'userSpecifiedTag'

      tag: 'api-v$(major).$(incremental)'

      title: 'API Release v$(major).$(incremental)'

      releaseNotesFilePath: '$(System.DefaultWorkingDirectory)/WebApi/InWords.WebApi/InWords.WebApi/releasenotes.md'

      assets: '$(Build.ArtifactStagingDirectory)/linux64/linux64.zip'

      addChangeLog: false