МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Управление корпоративными информационными процессами
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему Методы и модели интеграции мобильного приложения с КИС предприятия

Обучающийся Н. Б. Абдуллоев
(Инициалы Фамилия) (личная подпись)

Научный руководитель к.п.н., доцент, О. А. Крайнова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Оглавление

Введение 3
Глава 1. Анализ интеграции корпоративных информационных систем 6
1.1 Исследование предметной области 6
1.2 Подходы построения архитектуры КИС 13
1.3 Анализ существующих методов интеграции информационных систем 18
Глава 2. Методы разработки и интеграции мобильных приложений 29
2.1 Технологии разработки мобильных приложений 29
2.2 Модели и методологии разработки приложений 39
2.3 Методы проектирования пользовательского интерфейса 50
Глава 3. Интеграция мобильного приложения к КИС 58
3.1 Разработка Web-сервиса (API) 58
3.2 Разработка мобильного приложения 64
3.3 Интеграция мобильного приложения с КИС 68
Глава 4. Анализ эффективности интеграции 73
4.1 Результат разработки и интеграции мобильного приложения 73
4.2 Расчет эффективности интеграции разработанного мобильного приложения 76
Заключение 80
Список используемой литературы 81
Приложения А 84

Введение

Мобильные устройства и планшеты – это инструменты для получения разнообразной информации, используемые для решения повседневных задач. Сейчас эти гаджеты используются не только для общения, но и для решения рабочих задач. Для многих сфер деятельности, мобильные устройства стали очень важным каналом продаж, маркетинга, продвижения. Мобильный доступ ускоряет получение нужной информации, отправки отчетов, отслеживание за ходом каких-либо бизнес-процессов.

В современном мире очень трудно представить деятельность компаний без информационных систем. Из-за специфики стоящих задач приходится использовать различные информационные системы отвечает за выполнения определенных бизнес-процессов. Одна информационная система отвечает за введение бухгалтерского учета, другая за создание отчетов.

Таким образом, в компании увеличиваются количество информационных систем, отличающихся по своему типу, функционалу и других важных аспектов. В процессе увеличения масштаба, функциональных возможностей, информационные системы (далее ИС) сталкиваются с проблемой интеграции с другими ИС. Возникают ситуации, когда несколько ИС становятся частью одного бизнес-процесса.

Основная задача интеграции – это обеспечение эффективности, надежности и безопасный обмен между различными ИС, изначально не предназначенными для обмена с другими информационными системами.

По состоянию на первый квартал 2023 года Android остается самой популярной операционной системой в мире с долей рынка 71,62%, а за ней следует ее основной конкурент iOS с долей 27,73% [1]. Нужно принимать во внимание, разнообразность мобильных устройств при разработке корпоративного мобильного приложения, чтобы приложение одинаково функционировало на всех устройствах [2].

Актуальность темы исследования обусловлена необходимостью разработки модели интеграции мобильного приложения с корпоративной информационной системой «Колледж 24».

Объектом исследования магистерской диссертации является корпоративная информационная система (далее КИС) «Колледж 24».

Предметом исследования магистерской диссертации является разработка модели интеграции (мобильного приложения, интеграция).

Цель работы является анализ существующих методов разработки, интеграции мобильных приложений и разработка модели интеграции с КИС.

Для достижения цели, необходимо решить следующие задачи:

- Проанализировать архитектуру КИС;
- Проанализировать существующие методы интеграции информационных систем;
- Проанализировать существующие способы интеграции информационных систем с мобильными приложениями;
- Разработать модель интеграции мобильного приложения с КИС «Колледж 24»;
- Разработать и интегрировать мобильное приложение с КИС «Колледж 24»;
- Проанализировать результат интеграции мобильного приложения;
- Рассчитать модель эффективности разработанной системы.

Гипотеза исследования: разработанная на основе предлагаемых в диссертационном исследовании моделей, разработанное мобильное приложение и интеграция с корпоративной информационной системой «Колледж 24», обеспечит повышение эффективности взаимодействия сотрудников и студентов в учебном процессе.

Методы исследования: анализ архитектуры организации, анализ существующих интеграционных решений, методологии разработки программных обеспечений.

Научная новизна исследования заключается в разработке модели

интеграции мобильного приложения с КИС на основе анализа существующих методов и моделей разработки.

Практическая значимость исследования заключается в возможности применения предлагаемой модели в других корпоративных информационных систем.

На защиту выносятся:

— модель интеграции мобильного приложения с корпоративной информационной системой;
— результаты апробации и оценки эффективности разработанной системы.

Работа изложена на 84 страницах и включает 48 рисунка, 8 таблиц, 30 источников.

Глава 1. Анализ интеграции корпоративных информационных систем

1.1 Исследование предметной области

Во время предпроектного исследования было составлено описание предметной области решаемой задачи.

В современных бизнес-процессах используется разные ИС (планирование, транспортировка, бухгалтерия, учет персонала, разные прикладные и специализированные программные обеспечения для решения различных задач). Наличие множества ИС в организации является вполне нормальным явлением, т.к. каждая информационная система имеет свою функциональность. По мерее увеличения числа различных программных обеспечений появляются проблемы обмена между разными информационными системами. Интеграция между ИС обеспечивает работу бизнес-приложений с едиными точками входа и выхода.

Предметной областью исследования является Государственное бюджетное профессиональное образовательное учреждение Самарской области «Тольяттинский химико-технологический колледж».

Основная задача состоит в интеграции мобильного приложения к КИС и создание единой точки интеграции ИС с различными приложениями.

Организация имеет автоматизированную информационную систему «Колледж 24». Помимо основной информационной системы в образовательном учреждении существует еще множества других прикладных программных обеспечений.

Функциональная область корпоративной информационной системы «Колледж 24» состоит из следующих процессов:

— Администрирование;

— Новости | Мероприятия;

— Отдел кадров;

— Пользователи;

— Успеваемость;

— Расписания;

— Учебный процесс;

— Учебные материалы;

— Бухгалтерия;

— Документооборот;

— Приемная комиссия;

— Справки, заявления;

— Трудоустройство;

— Электронная библиотека;

— Настройки системы;

— Полезная информация;

— Техническая поддержка.

Исходя из организационной структуры образовательного учреждения были распределены роли пользователей в информационной системе. Каждая функция распределена по отделам. У каждого отдела свой пользовательский интерфейс с предопределенными функциями. Далее рассмотрим список отделов с уровнями доступа:

— Администраторы;

— Служба финансового обеспечения;

— Служба маркетинга образовательных услуг;

— Служба организации образовательной деятельности;

— Преподаватели;

— Студенты;

— Родители.

Рассмотрим каждую службу ролей по отдельности.

Администраторы

Администраторы в системе имеют доступ ко всему функционалу системы для анализа данных, добавления и изменения основных функций и

управлению ролями. На рисунке 1, представлен пользовательский интерфейс для администраторов.

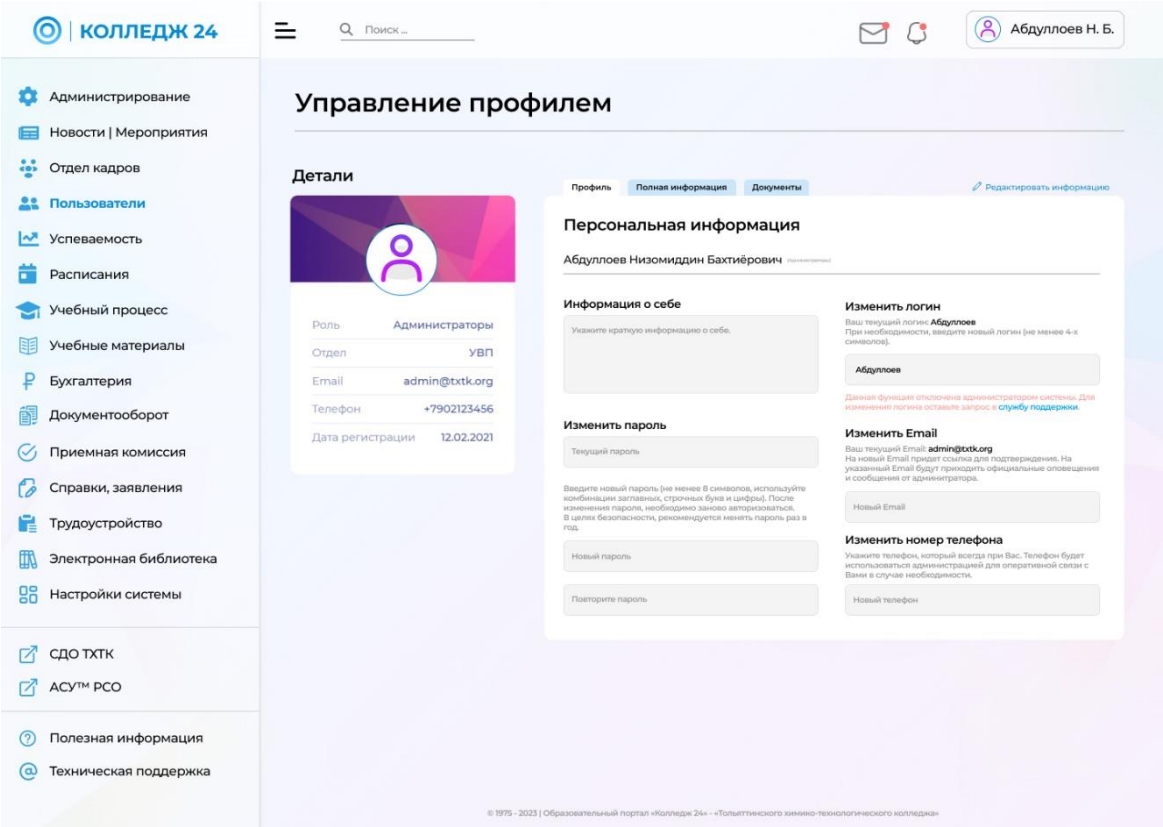

Рисунок 1 - Пользовательский интерфейс «Администраторы»

Служба финансового обеспечения (СФО)

В данную службу входят заместитель директора по финансово-экономической работе, бухгалтерия, отдел кадров. На рисунке 2. представлен пользовательский интерфейс для службы финансового обеспечения (СФО).

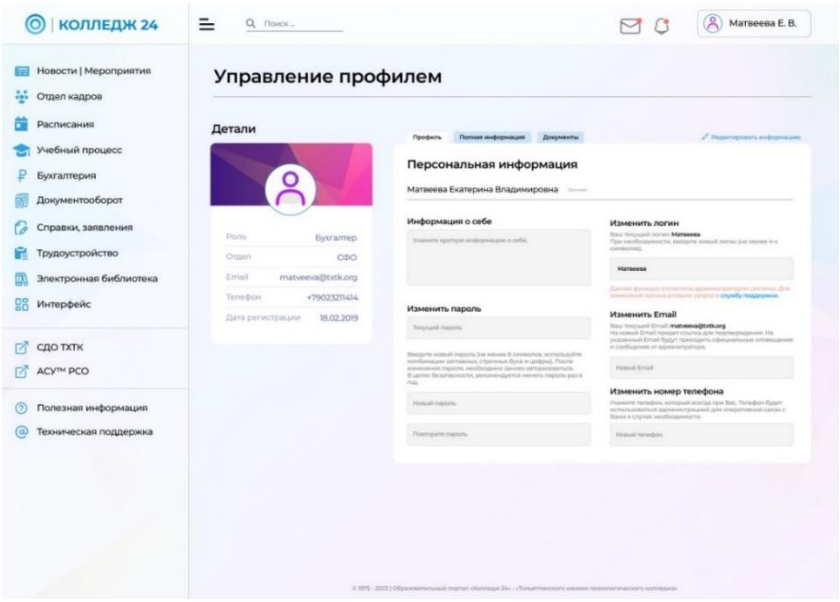

Рисунок 2 - Пользовательский интерфейс «СФО»

Служба маркетинга образовательных услуг (СМО)

В данной службе входят центр продвижения образовательных услуг, центр профориентации, центр содействия трудоустройству выпускников. На рисунке 3. представлен пользовательский интерфейс для службы маркетинга образовательных услуг (СМО).

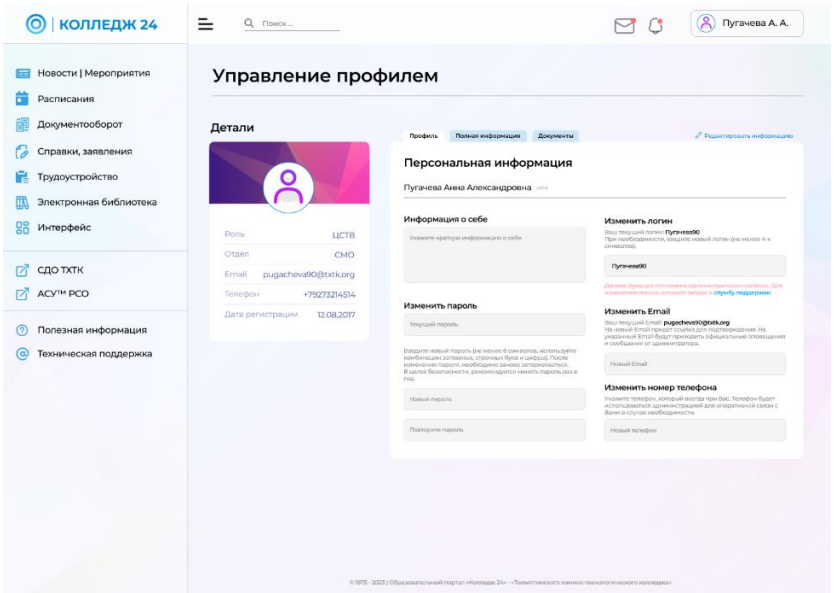

Рисунок 3 - Пользовательский интерфейс «СМО»

Служба организации образовательной деятельности (СООД)

В данную службу входят заместитель директора по учебной работе, приемная комиссия, очное отделение, заочное отделение, отдел организации практической подготовки, методическая служба, предметно-цикловые комиссии, центр непрерывного профессионального образования. На рисунке 4. представлен пользовательский интерфейс для службы организации образовательной деятельности (СООД).

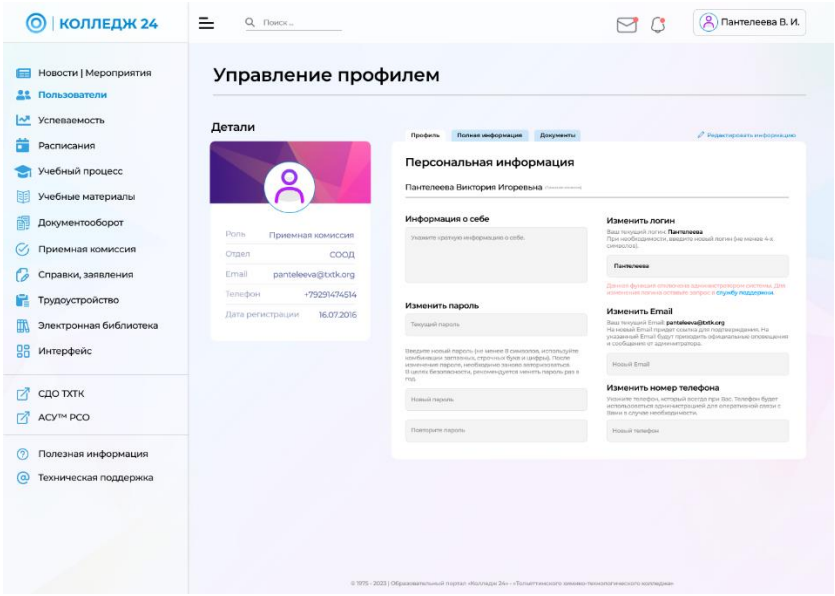

Рисунок 4 - Пользовательский интерфейс «СООД»

Преподаватели

В корпоративной информационной системе роль «Преподавателя» имеет свои особые функции, такие как:

— Классное руководство;

— Заполнение журнала посещаемости;

— Создание домашних заданий;

— Загрузка учебно-методических материалов;

— Выставление оценок по заданиям.

На рисунке 5. представлен пользовательский интерфейс для преподавателей.

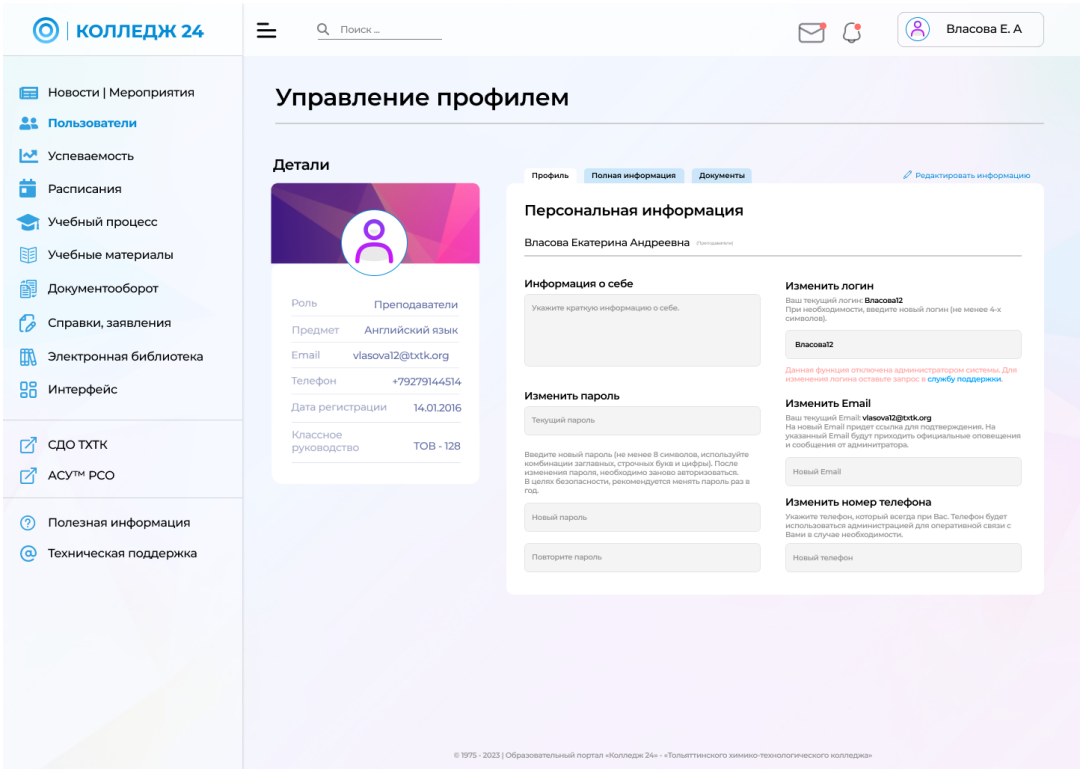

Рисунок 5 - Пользовательский интерфейс «Преподаватели»

Студенты

В корпоративной информационной системе роль «Студенты» имеет свои особые функции, такие как:

— Выполнение домашних задач;

— Просмотр академических задолженностей;

— Общение в общем чате группы;

— Получение доступа к расписанию своей группы.

— Запрос справок.

На рисунке 6, представлен пользовательский интерфейс для студентов.

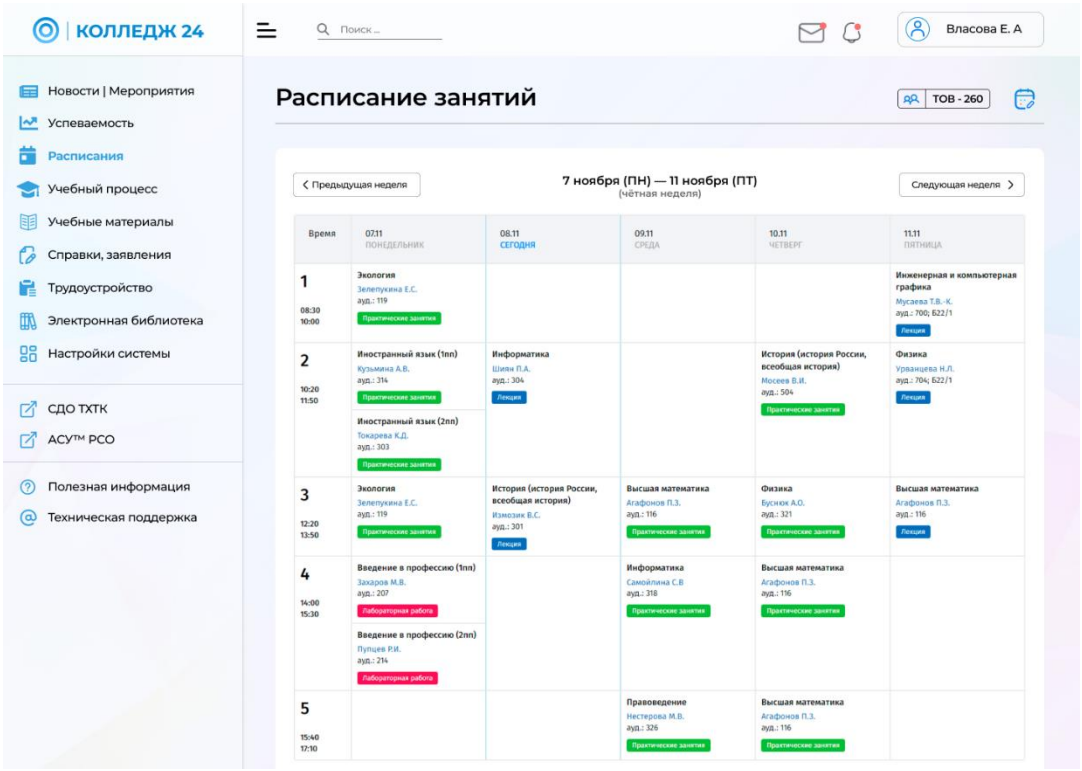

Рисунок 6 - Пользовательский интерфейс «Студенты»

Родители

Родители также имеют доступ к системе для отслеживания за ходом учебы студента(ов):

— Отчет посещаемости;

— Отчет по выполненным заданиям;

— Отчет по задолженностям;

— Общение с классным руководителем.

На рисунке 7. представлен пользовательский интерфейс для группы «Родители».

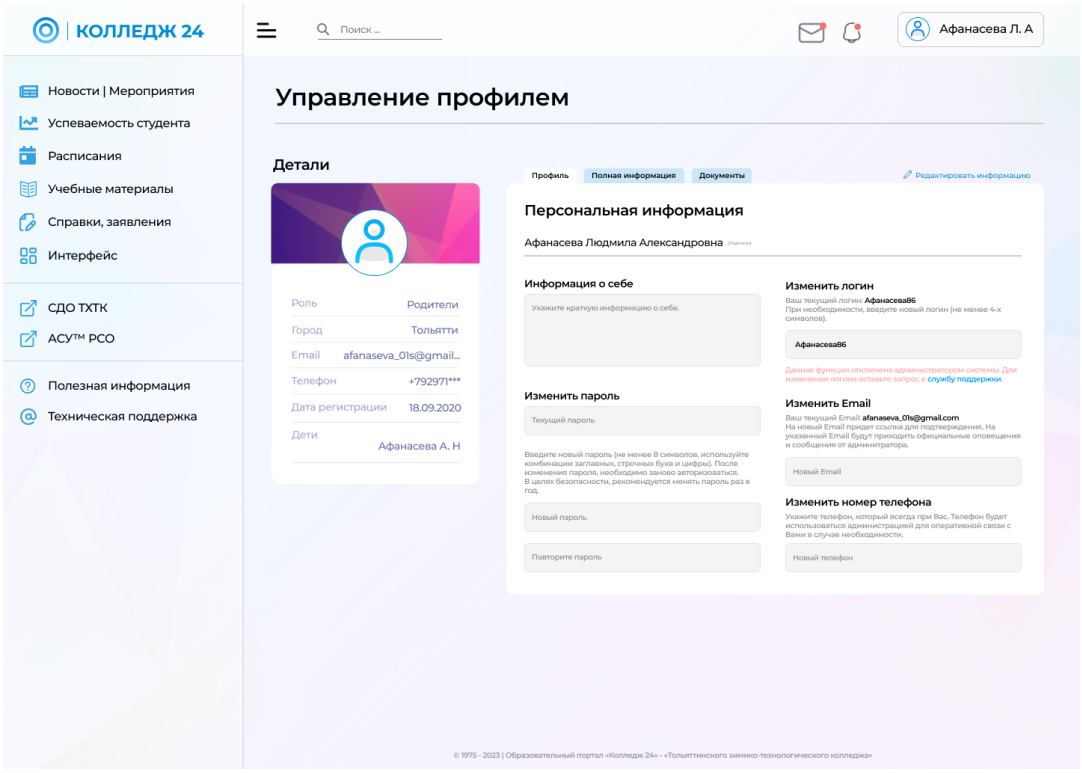

Рисунок 7 - Пользовательский интерфейс «Родители»

Исходя из вышеперечисленных ролей в корпоративной информационной системе «Колледж 24», можно подчеркнуть, что у каждого пользователя есть свои функциональные особенности.

Информационная система развернута на сервере колледжа и за обслуживание отвечает «Отдел технического и информационного обеспечения».

1.2 Подходы построения архитектуры КИС

В современном обществе, роль информационных технологий стремительно становятся основным техническим укладом. Сегодня ни одна современная компания не может эффективно управлять своей деятельностью без использования информационных технологий, а также надежной работой информационных систем.

Корпоративные информационные системы эффективно работают с разными типами данных при этом создавая цепочку новых информационных ресурсов для решения различных уровней задач. «Управленческая информация – это не только документооборот и финансы, это информация о структуре компании, бизнес-процессах, которые распределяют обязанности, задачи, цели бизнеса, информация обо всем, что влияет на работу компании».

Сейчас информационные системы отвечают не только за «технологичность», это и инструмент, без которого работа компании может остановиться, даже маленькие информационные сбои могут повлечь за собой огромные финансовые потери.

Под информационными технологиями в компаниях обычно понимают набор информационных систем, обеспечивающих поддержку и автоматизацию существующих бизнес-процессов [3].

«Информационные технологии – это система организационных структур, обеспечивающих функционирование и развитие информационного пространства предприятия и средств информационного взаимодействия. Основу информационных технологий составляет ИТ-инфраструктура».

«Техническое обслуживание – это комплекс мер программно-технического уровня, осуществляемых на этапе производственной эксплуатации и направленных на обеспечение требуемой надежности и эффективности функционирования информационной системы».

Можно выделить следующие задачи, решаемые ИТ-отделом:

— предотвращение и устранение сбоев;

— мониторинг работоспособности информационных систем;

— надежность ИТ-инфраструктуры компании;

— обеспечение информационной безопасности;

— сокращение расходов на поддержание информационных систем;

— обслуживание оборудования.

Техническая архитектура компании (Enterprise Technical Architecture, ETA) – совокупность всех программных и аппаратных средств, которые

обеспечивают функционирование разных программных продуктов [4]. Под технической архитектурой компании можно описать инфраструктуру компании, которая включает в себя:

— инфраструктуру компании;

— программное обеспечение;

— информационную безопасность;

— управление инфраструктурой.

Последовательно архитектуру компании, можно описать следующим образом (она отображена на рисунке 8):

— миссия, стратегия, цели и задачи;

— бизнес-архитектура (as is и to be);

— системная архитектура (as is и to be);

— план миграции.

Рисунок 8 – Циклическое развитие архитектуры предприятия

Как показано на рисунке, план миграции не замораживает бизнес-процесс.

Корпоративная информационная система – важный элемент современной информационной структуры компании, т.к. компании, которые

обладают высокой мерой сложности, подразделений и направлений деятельности в большинстве случаев нуждаются в корпоративной информационной системы с расширенным функционалом.

Корпоративная информационная система (КИС) – это комплекс программно-аппаратных средств, обеспечивающих бизнес-процессы компании [3].

Основные признаки КИС можно выделить следующем образом:

— соответствие информационным и управленческим потребностям компании;
— интегрированность;
— открытость;
— масштабируемость.

Информационную систему также называют корпоративной, если она охватывает необходимые сферы управления компании.

Распространенные типы КИС [6]:

— CRP (Capacity Requirements Planning) – системы, реализующие основные функции управления производством.
— FRP (Finance Requirements Planning) – системы, реализующие только технологии планирования и бюджетирования.
— MRP (Material Requirements Planning) – системы, специально разрабатываемые для нужд управления материальными ресурсами, в первую очередь – снабжением.
— MRP-II (Manufacturing Resources Planning) – комплексные системы финансового планирования и управления производством.
— MPS (Master Planning Shedule) – системы, ориентированные на большинство видов планирования, не только финансового, но и производственного, планирования продаж и т. д.

— CRM (Customer Relationship Management) – системы, ориентированные не только на обслуживание покупателя в связи с товаром, но и на любой тип клиентского обслуживания.
— SCM (Supply Chain Management) – логистические системы.
— ERP (Enterprise Resources Planning) – комплексные системы, реализующие большинство бизнес-процессов без выраженной доминанты какого-либо направления, но с возможностью «точной настройки» под нужды конкретного предприятия. Как правило, учитывают возможность как сквозного, так и оперативного контроля, что делает их исключительно удобными для использования топ-менеджментом. В настоящее время – наиболее распространенный и востребованный тип КИС.

Корпоративная информационная система «ТХТК» состоит из следующих информационных систем:

— «Колледж 24» - комплексное решение для управления деятельности ОО СПО;
— «Абитуриент 2.0» - веб-приложение для приема студентов;
— «КиберДиплом» - программный продукт для интенсивного ввода данных и быстрого заполнения бланков различных документов в образовательном учреждении;
— «1С Бухгалтерия» - профессиональный инструмент для ведения бухгалтерского учета;
— «Контроль» - система автоматического контроля учета посещаемости;
— «СБИС» - система управления персоналом;
— «10-Страйк Инвентаризация» - программный продукт для инвентаризации оборудований, программных обеспечений, лицензии и т.д.;
— «СДО ТХТК» - система дистанционного обучения.

Все вышеперечисленное программное обеспечение не имеет между собой интеграционных решений (либо частично имеют). Каждый раз приходится вручную из одной информационной системы переносить в другую, менять формат информации в читаемый для каждой информационной системы.

1.3 Анализ существующих методов интеграции информационных систем

Понятие интеграции в информационных системах – это автоматизированное взаимодействие между различными информационными системами и сервисами [7]. Простыми словами можно объяснить интеграцию как перемещение данных с одного места в другое или объединение частей в единое целое. Даже если мы переносим какие-то небольшие данные с одного места в другое – это тоже считается интеграцией.

Основная проблема заключается в том, что когда данные находятся в другой независимой информационной системе, то получить даже маленькие фрагменты бывает очень трудно т.к. у каждой системы свой алгоритм работы, своя информационная база. Еще важным нюансом можно отметить – большие данные, тут уже каждый раз перенести/получить данные с различных систем будем трудоемким и займет очень много ресурсов и требует много времени исходя из объема данных. В работе были проанализированы распространенные методы интеграции информационных систем.

Первоначально рассмотрим методы, определение задачи и основные направления их применения.

Enterprise Application Integration (EAI).

Данные приложения способны использовать разные технологии и оставаться независимо управляемыми.

EAI является технологией, при помощи которой организация достигает централизации и оптимизации интеграции корпоративных приложений,

используя, как правило, подходящие формы технологии оперативной доставки информации (push technology), которая управляется внешними событиями (event-driven) [5].

Выделяют две основных модели взаимодействия:

— Модель Point-to-point;

— Модель Hub-and-spoke.

Первую модель интеграции еще называют «точка-точка», которая отображена на рисунке 9. Данная модель относиться к прямому каналу связи между интегрируемыми системами напрямую без посредников и узлов.

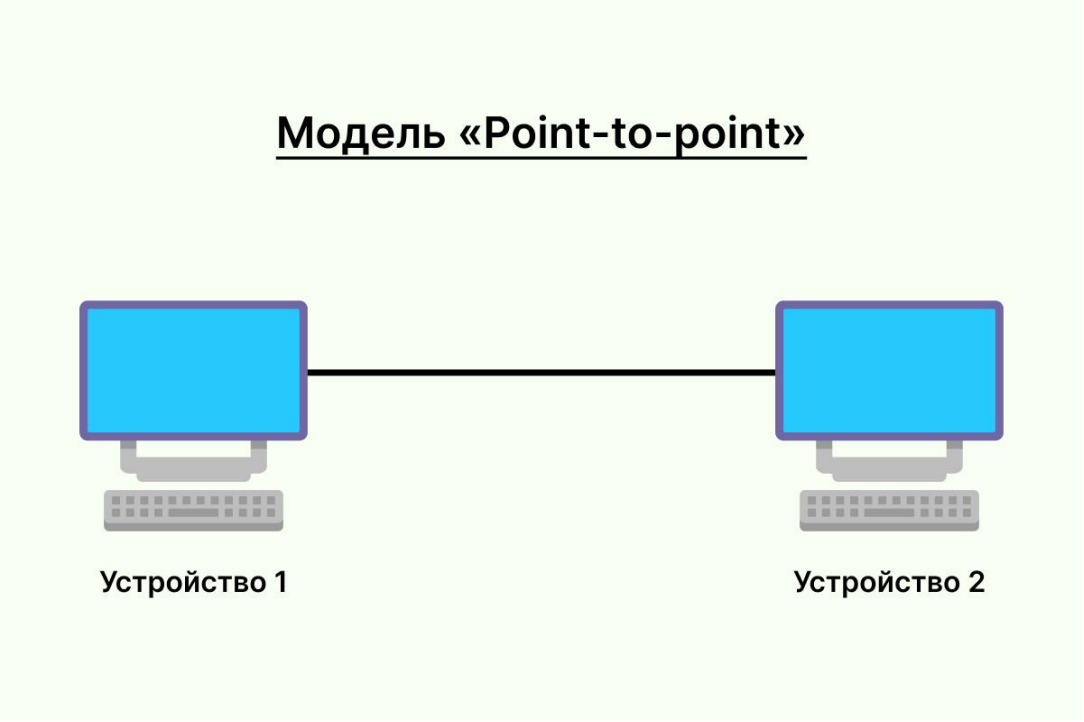

Рисунок 9 – Модель «Point-to-point»

Основное достоинство данной модели — это отсутствие необходимости применения каких-либо программных продуктов, простота и прозрачность системы.

Основные недостатки данной модели:

— интегрированные системы должны использовать одинаковые методы взаимодействия и форматы данных;

— информационные системы увеличиваются в организации и появляется проблемы каждый раз проверять и поддерживать работоспособность связей.

Из-за этого и увеличивается стоимость на содержание каждых узлов.

Модель Hub-and-spoke или концентраторно-спицевая модель – это система очень популярна в компьютерных сетях. Даная модель очень похожа на модель «точка-точка», но основное отличие — это дополнительный компонент «концентратора», который служит центральным узлом для других систем. На рисунке 10 отображена узлы связи по модели Hub-and-spoke [5].

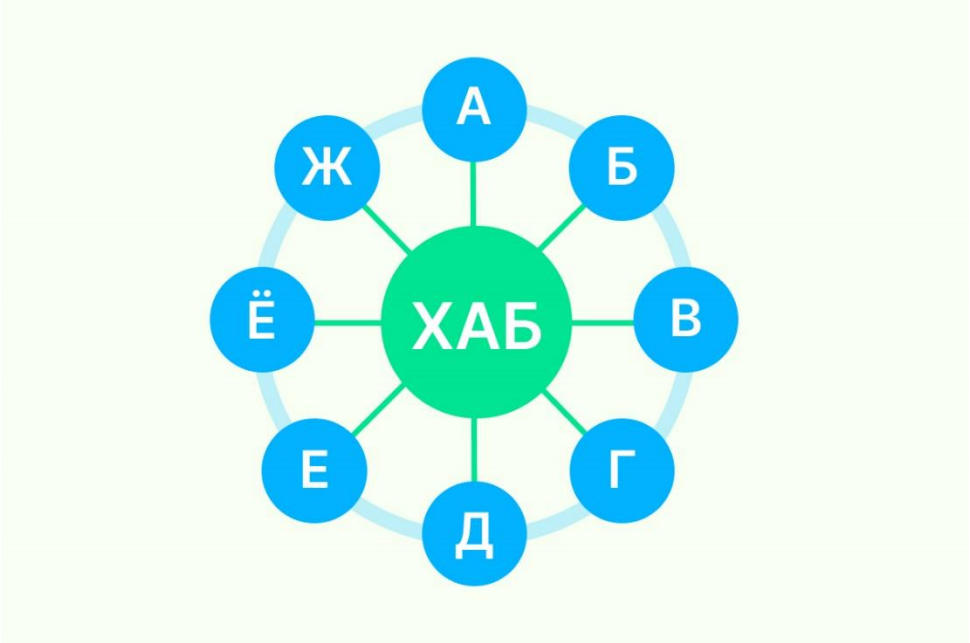

Рисунок 10 - Модель Hub-and-spoke

В модели Hub-and-spoke – концентратор отвечает за прием и передачу данных между системами.

Минус данной модели – это отказ центрального «хаба». Если хаб выйдет из строя, это повлияет на всю систему и связь между системами будет нарушена.

Модель Hub-and-spoke эффективно управляет потоками информации, тем не менее надо помнить о централизованных точек отказа и предпринимать стратегию по снижению рисков отказа.

Service Oriented Architecture (SOA)

Сервис-ориентированная архитектура (SOA) – концепция сервис-ориентированной архитектуры, предназначенная для решения вопросов интеграции информационной инфраструктуры компании за счет построения архитектуры, позволяющей интегрировать с максимальной гибкостью разнородные приложения [8].

Рисунок 11 – Сервис-ориентированная архитектура

Сервис предоставляет определенную бизнес-функцию, для обеспечения согласованной работы информационных систем (см. рисунок 11).

SOA не зависит от языков программирования, платформ или протокольных спецификаций, с помощью которых сервисы разрабатываются.

Основные принципы SOA являются:

— архитектура не связана с какими ни будь технологиями;
— независимость организации системы от используемых платформ;
— независимость от языков программирования;
— использование сервисов, независимых от конкретных приложений, с единообразными интерфейсами доступа к ним;
— организация сервисов как слабосвязанных компонент для построения систем.

Так же не маловажным аспектом SOA является высокая гибкость, которая достигается за счет возможности быстрой корректировки бизнес-логики - изменение, вносимое в бизнес-функцию, в итоге затронет все необходимые приложения.

Как правило, сервис-ориентированная архитектура реализуется с помощью веб-сервисов, что делает «функциональные строительные блоки доступными через стандартные интернет-протоколы».

Примером стандарта веб-службы является SOAP, что означает простой протокол доступа к объектам. В двух словах, SOAP «является спецификацией протокола обмена сообщениями для обмена структурированной информацией при реализации веб-сервисов в компьютерных сетях. Хотя поначалу SOAP не был хорошо принят, с 2003 года он приобрел большую популярность и становится все более широко используемым и принятым. Другие варианты реализации сервис-ориентированной архитектуры включают Jini, COBRA или REST.

Основные преимущества сервис-ориентированной архитектуры:

— Использование сервис-ориентированной архитектуры для создания повторно используемого кода. Это не только сокращает время, затрачиваемое на процесс разработки, но и избавляет от необходимости заново создавать код каждый раз, когда вам нужно

создать новый сервис или процесс. Сервис-ориентированная архитектура также позволяет использовать несколько языков программирования, поскольку все выполняется через центральный интерфейс.

— Использование сервис-ориентированной архитектуры для обеспечения взаимодействия: сервис-ориентированная архитектура обеспечивает стандартную форму связи, позволяющую различным системам и платформам функционировать независимо друг от друга. Благодаря такому взаимодействию сервис-ориентированная архитектура также может обходить брандмауэры, позволяя «компаниям совместно использовать службы, которые жизненно важны для работы».

— Использование сервис-ориентированной архитектуры для масштабируемости. Важно иметь возможность масштабировать бизнес в соответствии с потребностями клиента, однако определенные зависимости могут мешать этой масштабируемости. Использование сервис-ориентированной архитектуры сокращает взаимодействие между клиентом и службой, что обеспечивает большую масштабируемость.

SOAP — это упрощенный протокол, используемый для создания веб-API, обычно с помощью Extensible Markup Language (XML) [9].

SOAP является неотъемлемой частью сервис-ориентированной архитектуры (SOA) и спецификаций веб-сервисов.

Основные преимущества SOAP заключается в:

— Независимости от платформы и операционной системы. SOAP может передаваться по множеству протоколов, обеспечивая связь между приложениями с разными языками программирования как в Windows, так и в Linux.

— Работа по протоколу HTTP. Несмотря на то, что SOAP работает со многими различными протоколами, HTTP является протоколом по умолчанию, используемым веб-приложениями.
— Передача через различные сети и устройства безопасности. SOAP можно легко пройти через брандмауэры, где для других протоколов может потребоваться специальное приспособление.

А к недостаткам можно отнести:

— Отсутствие возможности передавать данные по ссылке. Это может вызвать проблемы с синхронизацией, если несколько копий одного и того же объекта передаются одновременно.
— Скорость. Структура данных SOAP основана на XML. XML в значительной степени удобочитаем, что упрощает понимание сообщения SOAP. Однако это также делает сообщения относительно большими по сравнению с архитектурой Common Object Request Broker (CORBA) и ее протоколом удаленного вызова процедур (RPC), который поддерживает двоичные данные. Из-за этого CORBA и RPC работают быстрее.
— Не такой гибкий, как другие методы. Хотя SOAP является гибким, новые методы, такие как архитектура RESTful, используют XML, нотацию объектов JavaScript, YAML или любой необходимый синтаксический анализатор, что делает их более гибкими, чем SOAP.

Основной конкурент SAOP является REST API. Везде можно наблюдать сравнение этих двух веб-сервисов.

RESTful API — это архитектурный стиль интерфейса прикладной программы (API), который использует HTTP-запросы для доступа и использования данных [10]. Эти данные можно использовать для типов данных GET, PUT, POST и DELETE, которые относятся к чтению, обновлению, созданию и удалению операций с ресурсами.

API для веб-сайта — это код, который позволяет двум программам взаимодействовать друг с другом. API описывает правильный способ

написания разработчиком программы, запрашивающей службы из операционной системы или другого приложения [5].

RESTful API, также называемый веб-службой RESTful или REST API, основан на передаче репрезентативного состояния (REST), который представляет собой архитектурный стиль и подход к обмену данными, часто используемые при разработке веб-сервисов. Наглядный пример обмена данными по протоколу RESTful API показана на рисунке 12.

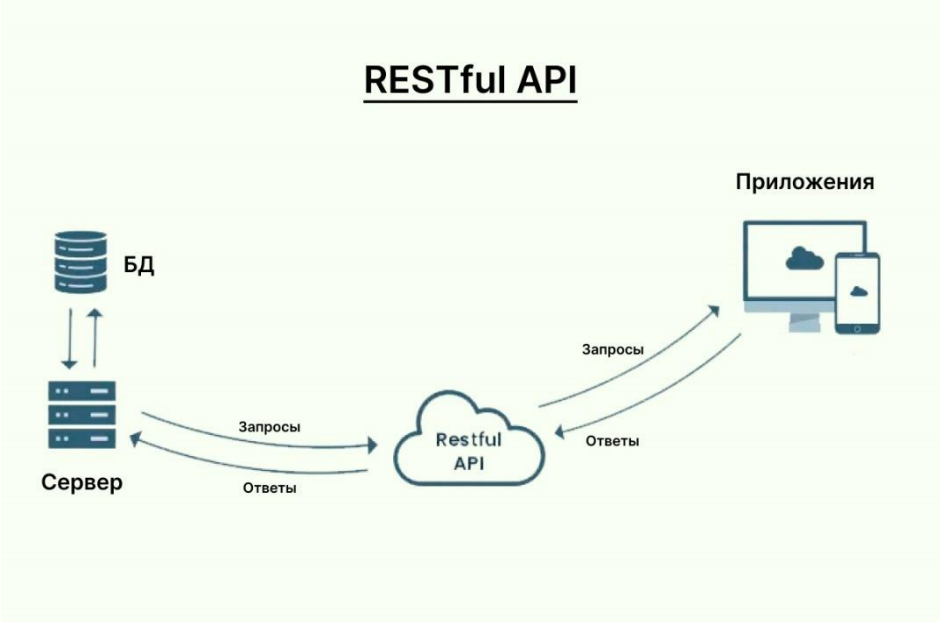

Рисунок 12 - Модель взаимодействия RESTful API

RESTful API использует команды для получения ресурсов. Состояние ресурса в любой момент времени называется представлением ресурса. RESTful API использует существующие методологии HTTP, определенные протоколом RFC 2616, такие как:

— GET, для получения ресурса;

— PUT, для изменения состояния или обновления ресурса, который может быть объектом, файлом или блоком;

— POST, для создания этого ресурса;

— DELETE, чтобы удалить его.

Форматы данных, поддерживаемые REST API, включают:

— application/json;

— application/xml;

— application/x-wbe+xml;

— application/x-www-form-urlencoded;

— multipart/form-data.

REST API популярен своей гибкостью, масштабируемостью, независимость и переносимостью. Архитектурный стиль REST использует особый подход к реализации клиента и сервера. Дело в том, что их можно делать независимо друг от друга и не нужно знать о другом.

Микросервисы – это сервисы, которые могут работать вместе и работать независимо. Эти сервисы взаимодействуют посредством сети [11]. Преимущества микросервисов заключается в их независимости. Архитектурный стиль данных сервисов – это разработка отдельного приложения в виде нескольких сервисов и каждый из них работают в собственном процессе взаимодействуя в упрощёнными механизмами такие как API HTTP или gRPC.

Микросервисы могут быть построены с учетом возможностей компании и развернуты независимо друг от друга с помощью автоматизированного процесса развертывания. Имеют минимальную централизованную схему управления данными сервисами, которые могут быть написаны на разных языках программирования и разных централизованных баз данных, что показано на рисунке 13.

Рисунок 13 - Структурная схема микросервисной архитектуры

Масштабировать микросервисы можно независимо друг от друга, т.к. масштабируется подсервисы, без масштабирования всего приложения.

ESB (Enterprise Services Bus) – сервисная шина предприятия, это интеграционная система, которая дает возможность обменяться информациями между разными информационными системами предприятия [12]. Обмен данными происходит через шину, которая использует разные протоколы и форматы, чтобы не менять логику интегрируемых систем. ESB обеспечивает преобразование сообщений в нужный формат, контроль, маршрутизацию, распределение нагрузки и безопасный обмен данными для конечных информационных систем, подробнее показана на рисунке 14.

Рисунок 14 - Взаимодействие приложений в ESB

В ESB каждое приложения взаимодействует только с интеграционной платформой. Если необходимо внести изменения в одной из систем, то на другие системы это не повлияет. Данные из разных приложений могут быть преобразованы в различные форматы, такие как JSON, CSV, XML и другие. ESB масштабируемая система, если в системе появится новое приложение, нагрузка будет распределена между приложениями.

Вывод по главе 1

В первой главе была исследована предметная область, которая заключалась в изучение организационной структуры ГБПОУ СО «ТХТК» и описана архитектуры организации. Из информационной структуры организации можно сделать вывод, что имеются различные информационные системы, которые не взаимодействуют (или частично взаимодействуют) друг с другом, а некоторые операции производятся вручную. Основная КИС, которая поддерживает рабочий процесс доступна только через браузер.

Также были проанализированы существующие методы интеграции информационных систем, которые предназначены для решения различных задач.

Глава 2. Методы разработки и интеграции мобильных приложений

2.1 Технологии разработки мобильных приложений

Разработка мобильных приложений — это процесс разработки программных приложений для мобильных устройств, включая мобильные телефоны и планшеты [13]. Типичное мобильное приложение использует сетевое подключение для работы с удаленными вычислительными ресурсами. Следовательно, в процессе разработки мобильного приложения требуется создать устанавливаемые пакеты программного обеспечения (включая код, двоичные файлы, активы и т. д.), развернуть серверные службы и протестировать приложение на целевых устройствах.

Для описания методов разработки мобильных приложений необходимо определить для каких платформ будет создано мобильное приложение.

Самыми популярными платформами на современном рынке, являются платформы iOS от компании Apple и Android от Google.

Разработка для iOS и разработка для Android имеют некоторые общие черты, но между ними имеется различия. Во-первых, каждый из них включает разные комплекты разработки программного обеспечения (SDK) и набор инструментов разработки. Во-вторых, Apple использует iOS исключительно для своих устройств, тогда как Android от Google доступен для других компаний, если они соответствуют требованиям платформы [14].

На этих двух платформах разработчики могут разрабатывать мобильные приложения для миллионов устройств.

Для разработки мобильных приложений необходимо выбрать подход, популярными их, которых являются:

— нативная разработка,

— кроссплатформенная разработка,

— гибридная разработка,

— прогрессивные веб-приложения.

Каждый из подходов имеют свои преимущества и недостатки, а при их рассмотрении необходимо учесть опыт пользователя, вычислительные ресурсы, встроенные функции, бюджет, срок разработки и ресурсы для поддержания жизненного цикла приложения.

Нативные мобильные приложения написаны на языке программирования и фреймворках, предоставленных владельцем платформы, для iOS это язык Swift, а для Android, Java и Kotlin [15]. Нативные мобильные приложения работают непосредственно в ОС устройства.

Основные преимущества нативных приложений:

— высокая производительность с точки зрения времени выполнения;

— прямой доступ к API устройства.

К недостаткам можно отнести:

— высокая затрата на разработку и поддержки приложения;

— различие кодовых баз для каждой платформы.

Кроссплатформенные мобильные приложения могут быть написаны на различных языках программирования и фреймворках, но они объединены в собственное приложение, которое будет работать в операционной системе устройства [16].

Основные преимущества кроссплатформенных приложений:

— единая кодовая база для нескольких платформ,

— простота разработки и обслуживания.

К недостаткам можно отнести:

— использование мостов и библиотек для нативных функций;

— ограничения производительности из-за моста.

Гибридные приложения или гибридные веб-приложения написаны на стандартных веб-технологиях, таких как JavaScript, CSS и HTML5. Они компилируются в установочные пакеты приложений. В отличие от собственных приложений, гибридные приложения работают в «веб-

контейнере», который обеспечивает среду выполнения браузера и мост для собственных API-интерфейсов устройств через Apache Cordova.

Основные преимущества кроссплатформенных приложений:

— веб-приложения и мобильные приложения используют одну и ту же кодовую базу;
— использование языков веб-разработки для разработки мобильных приложений.

К недостаткам можно отнести:

— более низкую производительность по сравнению с родными приложениями;
— ограниченную поддержку встроенных функций устройства.

Прогрессивные веб-приложения, это альтернативный подход к разработке традиционных мобильных приложений, при котором доставка в магазин приложений и установка приложений пропускаются [17]. Технически PWA — это веб-приложения, которые используют возможности браузера, такие как работа в автономном режиме, запуск фоновых процессов и добавление ссылки на главный экран устройства, чтобы обеспечить взаимодействие с пользователем, подобное приложению.

Основные преимущества кроссплатформенных приложений:

— одно и то же приложение доступно как для Интернета, так и для мобильных устройств;
— установка не требуется, доступ через URL.

К недостаткам можно отнести:

— ограниченную поддержку встроенных функций устройства;
— возможности приложения в зависимости от браузера.

Методология разработки мобильных приложений решает основные вопросы по набору команды разработчиков проекта, срокам и результатам.

Частый вопрос, который возникает при разработке мобильных приложений, это создание нативных приложений, гибридных приложений или

сайта с адаптивным дизайном для мобильных устройств. Как было упомянуто выше, нативные мобильные приложение предназначены для работы на конкретной мобильной среде, и они будут работать только на той ОС на котором был разработан приложение. Например, если это iOS, то необходимо создать на языке Swift или Objective-C [29].

Гибридные приложение или кроссплатформенные приложения работают одинаково на разных платформах. Важным преимуществом использование гибридных приложений является запуск на разных ОС без больших расходов. Если правильно разработать, то пользователи не смогут отличить, построено ли приложение с использованием гибридного или родного маршрута.

Кроссплатформенная разработка популярна для владельцев бизнеса, потому что за счет создания единой кодой базы для двух платформ Android и iOS можно сэкономить до 40% бюджета [18].

Для разработки кроссплатформенных приложений используются универсальные библиотеки и компоненты. Популярные фреймворки такие как: Flutter, React Native, Ionic, Xamarin, PhoneGap и другие. Сравнение технологий разработки приложений представлено на рисунке 15:

Рисунок 15 - Сравнение технологий для разработки мобильных приложений

Опишем подробно про популярные технологии разработки кроссплатформенных приложений.

Бесплатный фреймворк Flutter от компании Google с открытым исходным кодом для разработки пользовательского интерфейса мобильных приложений написан на языке Dart [19].

Flutter пользуется популярностью и многие разработчики и организации по всему миру называют его лучшим выбором для создания кроссплатформенных приложений. Приложения созданные на Flutter, полностью компилируются в нативный код. Такие компании как Alibaba, Tencent, Google, eBay и другие гиганты используют данный фреймворк для разработки собственных приложений.

Основным преимуществом данного фреймворка является быстрая разработка, красивые и выразительные нативно-скомпилированные приложения для Android (Material Design UI) и iOS (Cupertino UI) из единой кодовой базы. Основная информация по фреймворку Flutter представлена в таблице 1.

Таблица 1 – Информация по фреймворку Flutter

Дата выхода	2017
Лицензия	Open-source, BSD License
Языки разработки	Dart, C, C++
Доступ к Android Native API	Device API
Варианты развертывания приложения	мобильный, интернет, настольный, PWA
Компоненты пользовательского интерфейса	Богатый выбор элементов пользовательского интерфейса, виджеты для Material Design UI и Cupertino UI
MVC-архитектура	BLoC, Scoped Model, Vanilla
Интерфейс разработки	Командная строка (CLI), Android Studio
Форумы поддержки	GitHub, Сообщество разработчиков Flutter

React Native – это быстрорастущая платформа с открытым исходным кодом, который приобрел своё начало как внутренней хакатон-проект Facebook в 2013 г., после спустя 2 года она была выпущена, как кроссплатформенная среда разработки [20].

Такие гиганты как Facebook, Instagram, Tesla, Intuit, Bloomberg, Uber, Yahoo, Walmart и другие используют ее в своих проектах.

Преимущества в части элегантных пользовательских интерфейсов на различных платформах помогает разработчикам сократить затраты и время разработки. Основная информация по фреймворку React Native представлена в таблице 2.

Таблица 2 – Информация по фреймворку React Native

Дата выхода	2015
Лицензия	Open-source, MIT
Языки разработки	React, JavaScript, сторонние библиотеки
Доступ к Android Native API	JavaScript, сторонние библиотеки
Варианты развертывания приложения	мобильный, интернет, UWP
Компоненты пользовательского интерфейса	Native Widget
MVC-архитектура	Flux, Redux
Интерфейс разработки	Командная строка (CLI) React Native
Форумы поддержки	GitHub, StackOverflow, Dev Community

Ionic – это бесплатный фреймворк с открытым исходным кодом, позволяет создавать Android приложение, используя веб-технологии как HTML5, CSS3 и JavaScript [21]. Фреймворк для создания гибридных и интерактивных мобильных приложений и легко интегрируется с другими библиотеками, например, Angular и React, умеет работать с Bluetooth, Health Kit и проверкой подлинности по отпечатку пальца.

Огромная аудитория разработок, более 4 млн. созданных мобильный приложений, настольных и веб-приложений туристических агентств, ресторанов и других компаний. Основная информация по фреймворку представлена в таблице 3.

Таблица 3 - Информация по фреймворку Ionic

Дата выхода	2013
Лицензия	Open-source, MIT
Языки разработки	HTML5, CSS, JavaScript

Продолжение таблицы 3

Доступ к Android Native API	Библиотека плагинов с Cordova и Capacitor
Варианты развертывания приложения	мобильный, интернет, настольный, PWA
Компоненты пользовательского интерфейса	Основанные на стандартах компоненты пользовательского интерфейса
MVC-архитектура	AngularJS
Интерфейс разработки	Командная строка (CLI)
Форумы поддержки	Сообщество Ionic, Youtube-канал, Github, Twitter, форум разработчиков Ionic

Xamarin – фреймворк с открытым исходным кодом для создания приложений Android, iOS и Windows на базе технологии .NET.

Данный фреймворк имеет полный доступ ко всем возможностям SDK платформы и родному механизму создания UI.

«Xamarin имеет дружественную среду разработки, а его составная часть Xamarin.Forms позволяет создавать приложения с применением кода для пользовательского интерфейса, написанного на C# или XAML». Xamarin позволяет разработчикам писать всю бизнес-логику приложения, используя один язык программирования [22].

Преимущество данного фреймворка заключается в создании приложений сразу для 3х платформ, а код написанный на C# будет понятен разработчикам. Основная информация по фреймворку Xamarin представлена в таблице 4.

Таблица 4 - Информация по фреймворку Xamarin

Дата выхода	2011
Лицензия	Open-source, MIT
Языки разработки	C#, XAML, HTML5, CSS, JavaScript
Доступ к Android Native API	.NET
Варианты развертывания приложения	мобильный, интернет, настольный, PWA
Компоненты пользовательского интерфейса	Основанные на стандартах компоненты пользовательского интерфейса
MVC-архитектура	MVVM
Интерфейс разработки	Xamarin Forms, Microsoft Visual Studio, Android Studio
Форумы поддержки	StackOverflow, форумы разработчиков Xamarin

Сравнительный подход нативной и кроссплатформенной разработки мобильных приложений представлена в таблице 5.

Таблица 5 – Сравнение нативной и кроссплатформенной разработки

	Нативная разработка	Кроссплатформенная разработка
Технологии	Objective-C, iOS, SDK, Java, Android SDK	HTML, CSS, JavaScript, фреймворк для мобильной разработки
Публикация приложения	PlayMarket / AppStore	PlayMarket / AppStore
Скорость разработки	Медленная	Высокая
Цена разработки приложения	Высокая	Низкая
Цена поддержки приложения	Высокая	Низкая
Производительность	Высокая	Средняя
Доступ к API устройства	Да	Да
Где использовать	Во всех приложениях, где необходима производительность, график	В приложениях, которые не имеют высоких требований к производительности.

В ходе сравнения можно понять, что для решения поставленной задачи лучшим решением будет разработка кроссплатформенного приложения. После решения разработать кроссплатформенное приложение необходимо выбрать какой из фреймворков или платформ лучше использовать для решения данной задачи.

Проведем анализ платформ для разработки кроссплатформенных приложений и выберем самый подходящий для решения поставленной задачи. В процессе сравнения будут учитываться следующие характеристики:

— язык разработки,

— платформа,

— инструменты фронтенда,

— Open Source,

— производительность,

— сообщество.

Сравним вышеуказанные популярные платформы/фреймворки по различным критериям, которая представлена в таблице 6.

Таблица 6 - Сравнительная таблица анализа платформ/фреймворков для разработки мобильных приложений

	Flutter	React Native	Ionic	Xamarin
Язык разработки	Dart	React, JavaScript	JavaScript, HTML, CSS + Angular, React, Vue	C# + .NET
Платформа	Android, iOS, веб-приложений, Windows, macOS и Linux	Android, iOS, веб-приложений, Windows, macOS	Android, iOS, веб-приложений	Android, iOS, UWP
Инструменты фронтенда	Встроенные виджеты	Компоненты Native и Declarative UI	HTML, CSS, виджеты	Xamarin iOS/Android, или Xamarin.Forms
Open Source	Да	Да	Да, Платные пакеты	Да, Платные пакеты
Производительность	Очень высокая, близка к нативной	Очень высокая	Средняя	Высокая
Сообщество	Большое	Очень большое	Большое	Большое

На основе сравнительной таблицы выявлено, что лучшей платформой для разработки мобильного приложения для дальнейшей интеграции является платформа/фреймворк Flutter.

После выпуска первой версии Flutter, в основном применялось для мобильной разработки. Сейчас Flutter поддерживает разработку приложений на нескольких платформах, таких как Android, iOS, Web-приложения, Windows, Linux и MacOS. Также Flutter – это платформа для разработки, с поддержкой нескольких платформ одновременно. Для определенной задачи

используется два метода разработки под конкретную платформу: платформозависимые и кроссплатформенные методы разработки.

Платформозависимые приложение в Flutter – разработка целенаправленно под конкретную платформу, например, Android. При разработке платформозависимых приложений, в первую очередь основным преимуществом, является полный доступ ко встроенным функциям устройства, что обусловлено более высокой производительностью и скоростью приложений по сравнению с кроссплатформенным методом разработки.

Основным минусом платформозависимых приложений в Flutter является то, что разработка будет работать под конкретную операционную систему или потребуется много финансовых затрат на дополнительных разработчиков для других платформ. А также платформозависимые приложения сложно запускать на других платформах с одинаковым интерфейсом.

Кроссплатформенная разработка на Flutter, дает возможность разработать и использовать один язык программирования и единую кодовую базу для запуска на разных платформах (см. рисунок 16). Основным преимуществом Flutter в кроссплатформенной разработке является использование от 85% до 95% кодовой базы для других платформ в зависимости от специфики проекта [23].

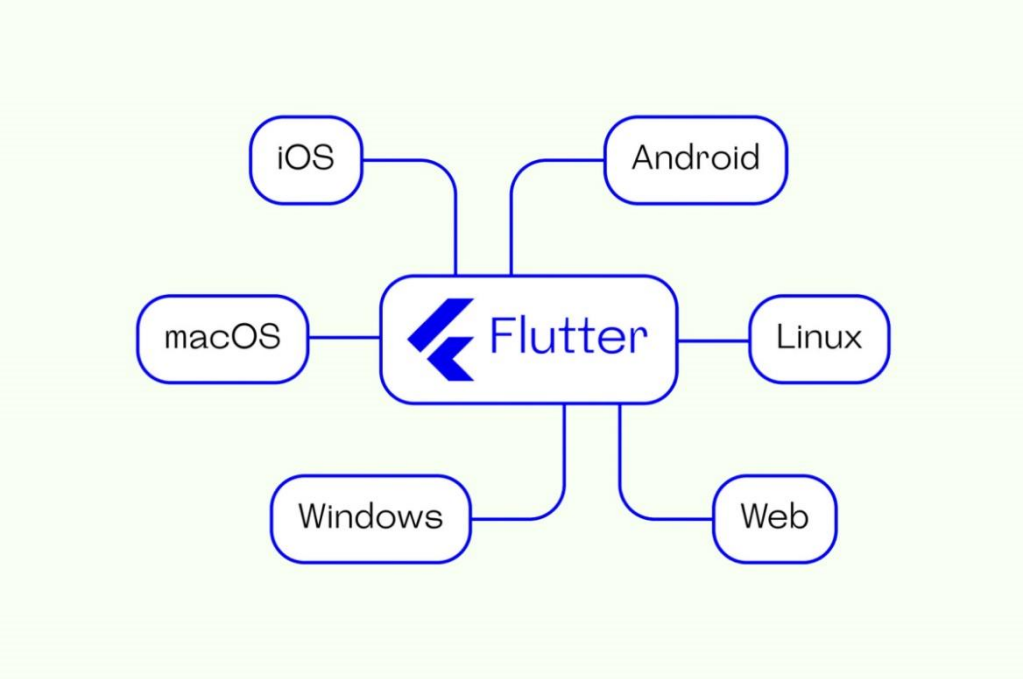

Рисунок 16 - Кроссплатформенная разработка на Flutter

В результате последних разработок в банковской сфере, ритейла и других бизнесов, выяснилось, что бизнес сэкономил затрат на разработку при использовании Flutter [24]:

— до 46% времени и стоимости на разработку,

— 70% на тестированиях,

— до 32% на дизайне,

— до 70% на дальнейшую поддержку,

— до 20% на добавление новых функций после запуска проекта.

Flutter поддерживается компанией Google, и постоянно обновляется, имеет активное сообщество, документацию. Компания Google активно проводит глобальные мероприятия, состязания и спонсирует разработчиков.

2.2 Модели и методологии разработки приложений

Рассмотрим классические модели разработки и современные модели разработки. Именно развитие классических моделей привело к появлению

гибких методологий разработки программных обеспечений, которые широко используется в наше время.

«Модель жизненного цикла программного обеспечения описывает какие этапы проходит программное обеспечения от рождения идеи до завершения использования, и что происходит с программным обеспечением на этих этапах» (см. рисунок 17).

Рисунок 17 – Модель жизненного цикла программного обеспечения

Методология разработки программного обеспечения – это набор методов по управлению разработкой программного обеспечения, набор практических правил и техник разработки [25].

Существует множества моделей разработки программных обеспечений, перечислим 5 самых распространённых моделей разработки, их описание, преимущества и недостатки:

— «Waterfall Model» (каскадная модель или «водопад»);
— «V-model» (V-образная модель, разработка через тестирование);
— «Incremental Model» (инкрементная модель);
— «Iterative Model» (итеративная или итерационная модель);
— «Spiral Model» (спиральная модель).

Каскадная модель (waterfall model) или «Водопад» является самым старым подходом к разработке программного обеспечения. Основная особенность данной модели, заключается в том, что разработка осуществляется поэтапно, а каждый следующий этап начинается поле того, как заканчивается предыдущий, т.е. невозможно перейти на следующий этап пока первый или предыдущий этап не закончен. Каскадная модель изображена на рисунке 18.

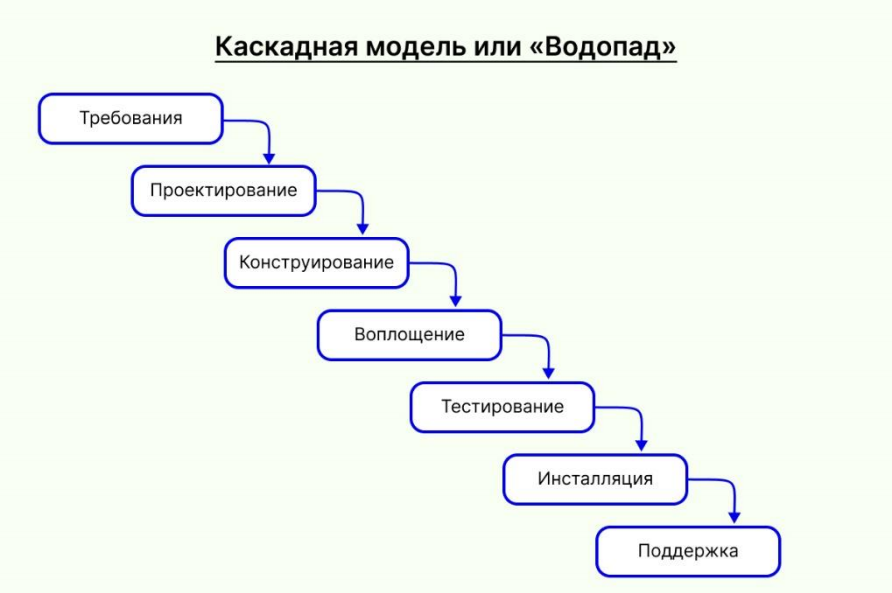

Рисунок 18 – Каскадная модель

Преимущества данной модели заключаются в следующем:

— простота в изучении,

— контроль каждого этапа,

— определение стоимости проекта в начале разработки.

Недостатки данной модели заключаются в следующем:

— нельзя вернуться назад,

— тестирование происходит на более поздних этапах,

— создается большое количество документации.

V-образная модель (V-Model) – это усовершенствованная каскадная модель. В данной модели усовершенствовали планирование тестирования, т.е. когда система будет готова прописывают, какие тесты будут приемочным для каждого этапа разработки, таким образом можно уже на стадии разработки выявить какие недостатки существуют в требованиях. Данная модель дает высокую надежность, например, используется для системы безопасности автомобиля, системы наблюдения за пациентами в клиниках, т.е. используется в таких проектах, где цена ошибки очень высока. V-образная модель отображена на рисунке 19.

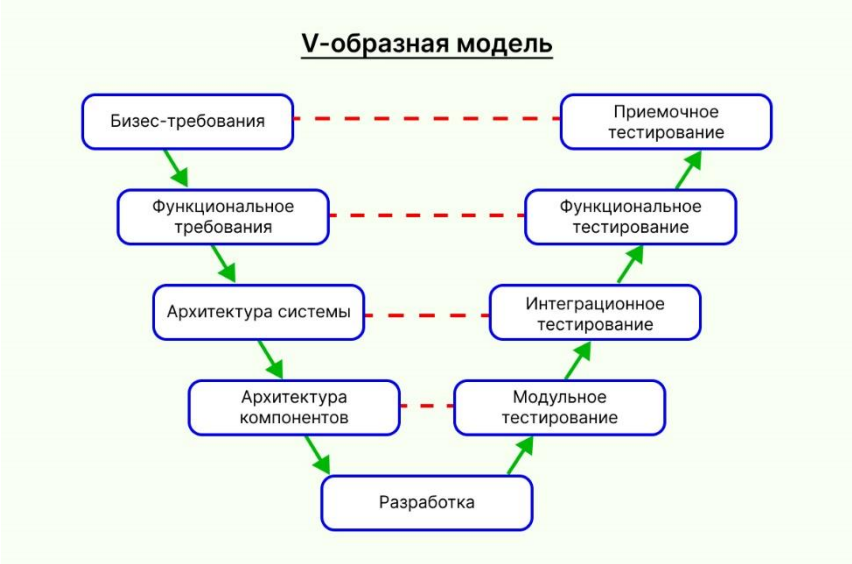

Рисунок 19 – V-образная модель

Преимущества данной модели заключается в следующем:

— простота в изучении,

— контроль каждого этапа,

— планирование тестов на ранних этапах,

— определение стоимости проекта в начале разработки.

Недостатки данной модели заключается в следующем:

— невозможности вернуться назад,

— каждая итерация ошибок дорого будет стоить,

— большое количество документации.

Инкрементная модель (Incremental Model)

Инкрементная модель – это метод, в котором проект проектируется разрабатывается и тестируется инкрементно, т.е. каждый раз с небольшими добавлениями до окончания разработки (см. рисунок 20). Включает в себя разработку и дальнейшую поддержку программного обеспечения.

Рисунок 20 – Инкрементная модель

Преимущества данной модели заключается в:

— рабочее приложение выходит на ранней стадии жизненного цикла продукта;

— небольшие итерации упрощают тестирование и внесение правок;

— гибкость. Изменить масштабы и требования проекта относительно менее затратно;

— каждая итерация — простая в управлении контрольная точка проекта.

Недостатки данной модели заключается в:

— каждая фаза итерации неподвижна;
— могут возникнуть проблемы относительно архитектуры системы, так как не все требования собраны заранее для всего жизненного цикла ПО.

Итеративная или итерационная модель (Iterative Model)

Основная суть данной модели создание базового функционала и постепенное его улучшение. Нет точного количества шагов, весь процесс разработки происходит в циклах. Очень мало внимания уделяется на отслеживание и результативность других функций. Главная цель – это создать рабочий прототип и добавлять функции в процессе работы циклов разработки. Итерационная модель ПО успешно реализована и пройдена, когда происходит полная проверка всех требований и проверка каждой версии продукта на каждом этапе в соответствии с заявленными требованиями.

Преимущества данной модели заключаются в следующем:

— создание программы происходит в начале жизненного цикла и достаточно быстро;
— гибкое управление, моделирование. Требования и направления изменяются легко и дешево;
— легко тестировать и производить отладку на небольших итерациях;
— любое повторение – этап легкий в своих повторениях.

Недостатки данной модели заключаются в следующем:

— возможны вопросы по архитектуре системы. Не каждые требования подходят для всего жизненного цикла разрабатываемого продукта;
— всякий этап итерации суровый, не встречающийся с предыдущим.

Спиральная модель (Spiral Model)

Это модель процесса разработки программного обеспечения с учетом рисков. Это комбинация модели водопада и итеративной модели. Spiral Model помогает внедрить элементы разработки программного обеспечения из нескольких моделей процессов для программного проекта на основе

уникальных шаблонов рисков, обеспечивая эффективный процесс разработки. Каждая фаза спиральной модели в разработке программного обеспечения начинается с определения цели проектирования и заканчивается тем, что клиент просматривает прогресс.

Наиболее важной особенностью спиральной модели является управление этими неизвестными рисками после начала проекта. Такое решение рисков легче осуществить, разработав прототип. Спиральная модель помогает справляться с рисками, предоставляя возможности для создания прототипа на каждом этапе разработки программного обеспечения. Схему спиральной модели можно посмотреть на рисунке 21.

Рисунок 21 – Спиральная модель

Преимущества данной модели заключаются в следующем:

— обработка рисков,

— подходит для больших проектов,

— гибкость в требованиях,

— удовлетворенность клиентов.

Недостатки данной модели заключаются в следующем:

— сложность,

— дорого,

— слишком сильно зависит от анализа рисков,

— сложность в управлении временем.

Перечислили 5 самых популярных методов разработки программных продуктов, которые до сих пор применяются для различных проектов. Развитие данных моделей привело к появлению гибких (гибридных) методологий разработки программных обеспечений, которые широко используется в наше время. Перечислим самые популярные методологии, которые используются в проектах повседневно.

Scrum - это методология гибкой разработки. Scrum относится к семейству Agile. Принцип Scrum это ценность командной работы, причем ставка делаются не только на действия сотрудников, но и на клиента, в интересах которого решается все поставленные задачи и участвует во всех этапах разработки. Коммуникация между клиентом и разработчиками дает точные результаты в процессе разработки.

Ценность данной методологии заключается во вовлечении заказчика в процесс разработки. На каждом этапе заказчик может внести корректировки и озвучить новые требования к разработке продукта. Благодаря принципам спринтов, небольшие компании могут обойтись без привлечения узких специалистов под каждые задачи компании.

Несмотря на простоту, Scrum имеет свои недостатки, т.к. правила Scrum четкие и жесткие. Есть временные циклы, которые нельзя нарушить в разработке продукта. По сути, в Scrum, каждый спринт — это риск, который в итоге может сформировать новые взгляды на поставленные задачи.

Обязательные принципы Scrum:

— работа короткими циклами (спринтами),

— гибкость,

— участие заказчика в разработке,

— взаимодействие команды.

Kanban – это методология, по сути это даже не методология а инструмент, который можно применить практически с любым подходом. Kanban, делает работку над проектом нагляднее, позволяя отслеживать выполнение отельных задач и легко контролировать загруженность каждого специалиста.

Чтобы контролировать выполнение задач, процесс визуализируют на доске, разделенной на несколько колонок. Все поступающие задачи, записываются на стикеры и размещаются в нужную колонку. Для каждой поступающей задачи можно выставить приоритеты исходя из важности задач.

Важное правило, которое придерживает Kanban, это ограничение количество стикеров-задач (см. рисунок 22). Точное количество определяется исходя из возможностей команды. Исполнитель не может взяться за следующую задачу, не раньше, чем закончить предыдущую. Ограничительные меры на практике получают иной подход, и отношение к незавершенным задачам. Если задача не была выполнена или имеет задержки, то благодаря Kanban-доске сразу будут заметны, не только разработчику, но и всему коллективу. Это стимулирует разработчика для более эффективного выполнения задач, т.к. одна задача может зависеть от другой, что позволяет разработчикам тесно сотрудничать, обсуждать возникающие вопросы во время выполнения задачи.

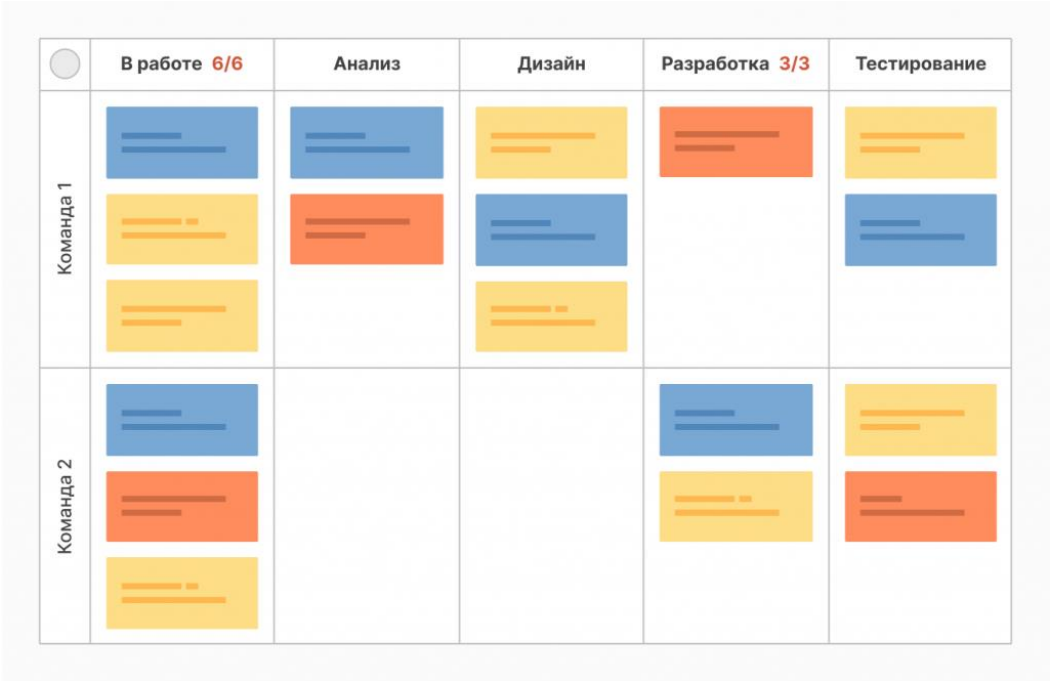

Рисунок 22 - Kanban-доска

Виртуальные Kanban-доски сейчас очень популярны. Они упрощают работу в отличие от стикеров на стене и доступны не только в офисе, но и в любой точке мира, идеальное решение если команда работает с разных городов и стран.

Сейчас самые популярные виртуальные Kanban-доски это:

— Trello,

— Kanbanchi,

— Битрикс24,

— Kanbanery,

— Taskify.

Среди многообразия виртуальных решений для каждой компании найдется, та, которая подходит под определенные задачи.

Agile – это философия, группа методологий, для улучшения производимого продукта с помощью повторяющихся циклов и постоянной обратной связью от заказчиков. Философия Agile характеризуется гибкостью, скоростью работы команды, прозрачностью рабочих процессов, постоянным

общением с заказчиком для оперативного реагирования на изменения в ходе разработки.

Чаще всего Agile используется для разработки программных продуктов, базируясь на итерациях фаз программирования и тестирования на протяжении полного цикла разрабатываемой продукции (см. рисунок 23).

Основные преимущества Agile:

— обнаружение и исправление ошибок,

— прозрачность,

— частые релизы,

— прогнозируемость стоимости и время,

— высокий уровень удовлетворенности заказчика.

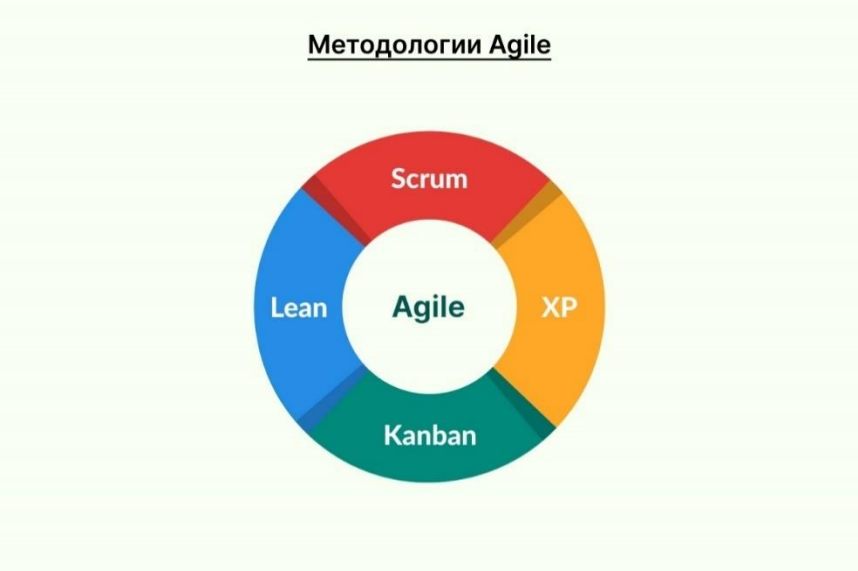

Рисунок 23 – Методология Agile

Существует множества фреймворков (методологий) для гибкой разработки, Kanban, Scrum, Бережливое производство (Learn) и Экстремальное программирование (XP).

Agile – незаменимый подход к управлению проектами, для достижения лучших результатов. Благодаря тесному сотрудничеству команды и заказчика конечный продукт будет удовлетворять всем принципам разработки.

2.3 Методы проектирования пользовательского интерфейса

Термин «пользовательский опыт» или часто используют синоним «пользовательский интерфейс» (UX) и «юзабилити» (UI), являются важными аспектами при проектирования различных программных обеспечений, визуализации дизайна и многих других дисциплин. Наиболее понятным термином считаются UI и UX, но на практике можно заметить UI/UX-дизайнер воспринимается как одна профессия. Интерфейс — это инструмент, а взаимодействие – более шире, чем интерфейс и описывает в формате диалога как «вопрос - ответ» или «запрос – результат» между пользователем и объектом [30].

Вид прототипа, который необходимо создать, зависит от проекта, условий и ограничений, т.е. что в конечном итоге будем тестировать, на каком уровне находиться проект, сколько ресурсов будет потрачено в процессе разработки в зависимости от нарисованного прототипа (см. рисунок 24).

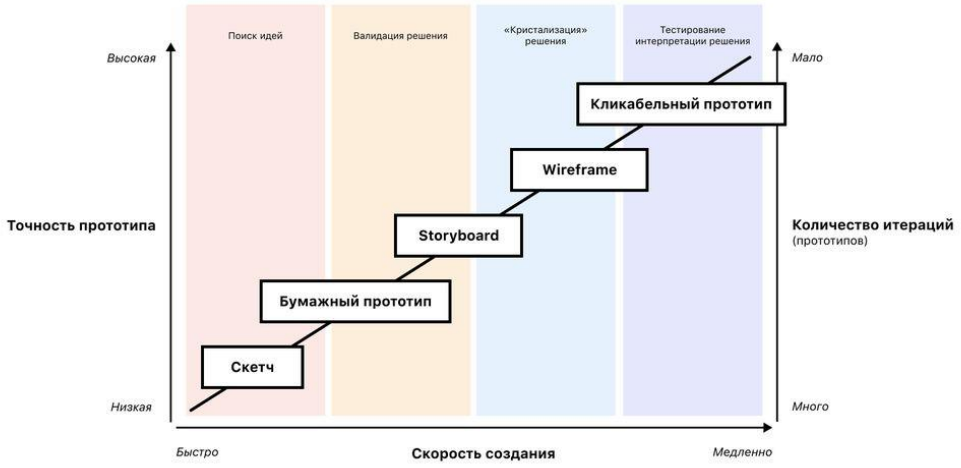

Рисунок 24 – Виды прототипирования

Существуют несколько видов прототипирования, такие как скетч, бумажный прототип, storyboard, wireframe, кликабельный прототип. Кратко опишем самые популярные виды прототипирования.

Скетч – это рисунки. Они выражают идеи и концепты с помощью визуальных образов. В скетчинге не требуется точно копировать форму предметов или рисовать каждую деталь по отдельности, а просто быстрый рисунок, который показывает концепт продукта.

Бумажный прототип – модель продукта, нарисованная или сделанная из нее. Бумажный прототип позволяет быстро и практически без затрат создать модель идеи, показать пользователю и быстро внести изменения если оно потребуется.

Wireframe – это концепция структуры дизайна интерфейса. Это, по сути, «Скелет» конечного программного продукта, который держит на себе все остальные части тела. Если пропустить этап создания «Скелета», то это добавляет лишнюю работу во время реализации дизайн-процесса или конечного продукта.

Кликабельный прототип – интерактивная схема конечного продукта, которое симулирует взаимодействия пользователя с интерфейсом. Данный вид прототипирования позволяет проверить юзабилити разрабатываемого программного продукта, перед началом написания кода, а значить, сократить сроки и бюджет на разработку.

Согласно исследованию от компании UXtools.co, самые популярные инструменты для создания прототипов в 2022 году отображены на рисунке 25:

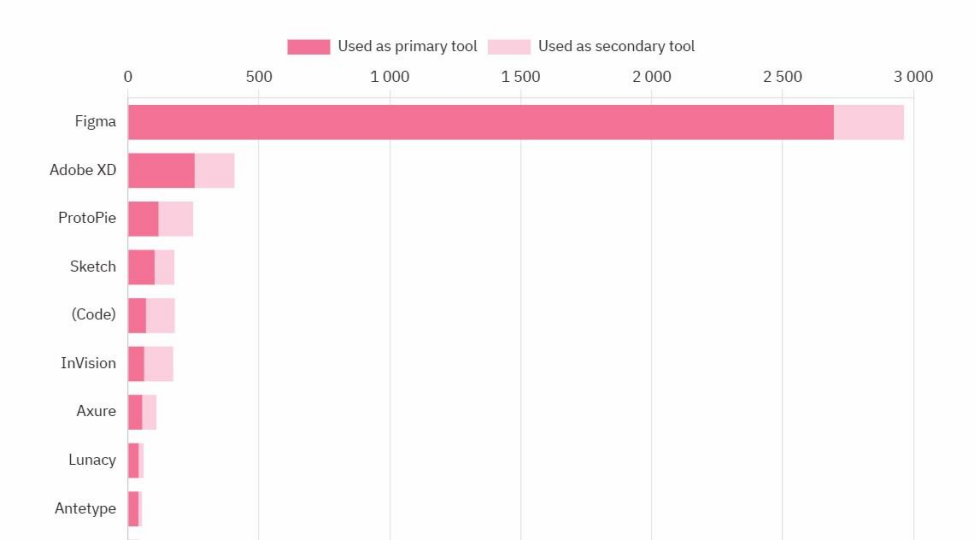

Рисунок 25 – Популярные инструменты для прототипирования

Рассмотрим первые 4 из них – стоимость, преимущества и недостатки.

Figma – это универсальный инструмент для создание прототипов и дизайнов. Данный инструмент решает такие задачи как, прототипирование, создание полноценного дизайна, создание презентаций (см. рисунок 26). Данный инструмент доступен для командной работы, это очень удобно, когда над проектом работают много разработчиков.

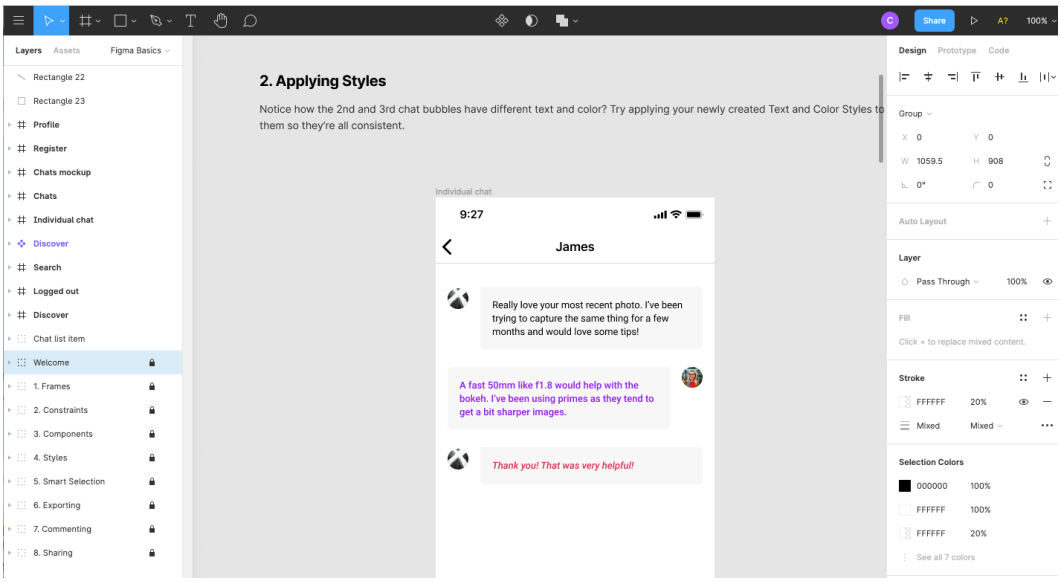

Рисунок 26 – Пользовательский интерфейс Figma

Доступ к прототипу осуществляется через браузер. Есть desktop приложение, но оно используется реже чем браузерная версия, на рисунке 27 отображен пользовательский интерфейс desktop версии. Все прототипы хранятся в облаке Figma, что делает инструмент очень удобным.

Основной минус данной системы, постоянное подключение к сети. Десктоп версию можно использовать без подключения к сети, но в конечном итоге для отправки проделанной работы с дополнениями необходимо подключиться к сети.

Adobe XD – разработка от компании Adobe, которая показала себя с лучшей стороны уже давно. Основной технологический процесс программного обеспечения, создание дизайна, прототипа, презентаций. В основном используется для больших команд, где требуется создавать проекты в большом масштабе.

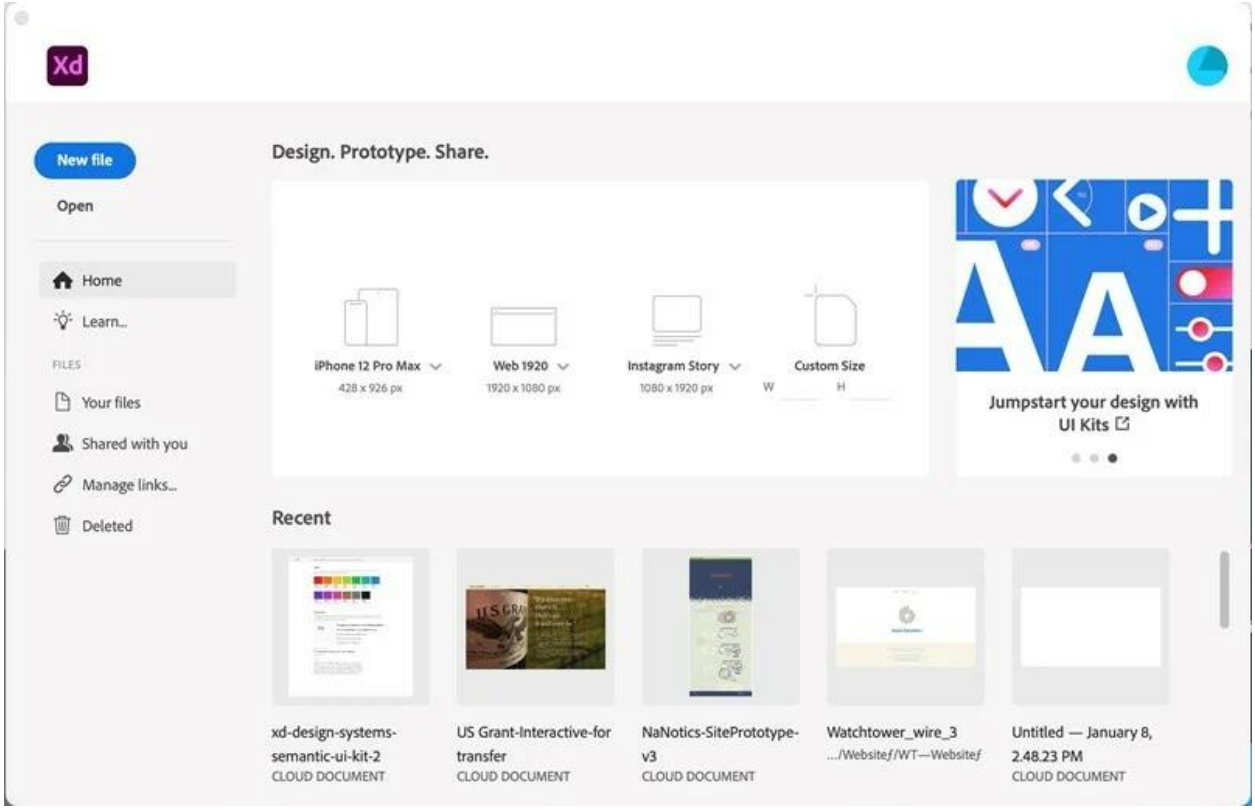

Рисунок 27 – Пользовательский интерфейс Adobe XD

Есть большая база готовых прототипов для наполнения контентом, можно импортировать множество форматов из различных графических программ. Из минусов можно выделить высокую нагруженность программы, что замедлят работу системы, частые обновления, которые вносят в себя технические сбои и тормозят рабочий процесс.

ProtoPie – программное обеспечение для создания интерактивных прототипов мобильных и десктопных приложений, сайтов и презентаций визуально неотличимых от настоящих. Программное обеспечения позволяет продемонстрировать дизайн-макет интерфейса проекта, презентовать заказчику, и провести тест. Позволяет использовать датчики в смартфонах такие как, камера, гироскоп, акселерометр, микрофон и др. Пользовательский интерфейс инструмента ProtoPie изображен на рисунке 28.

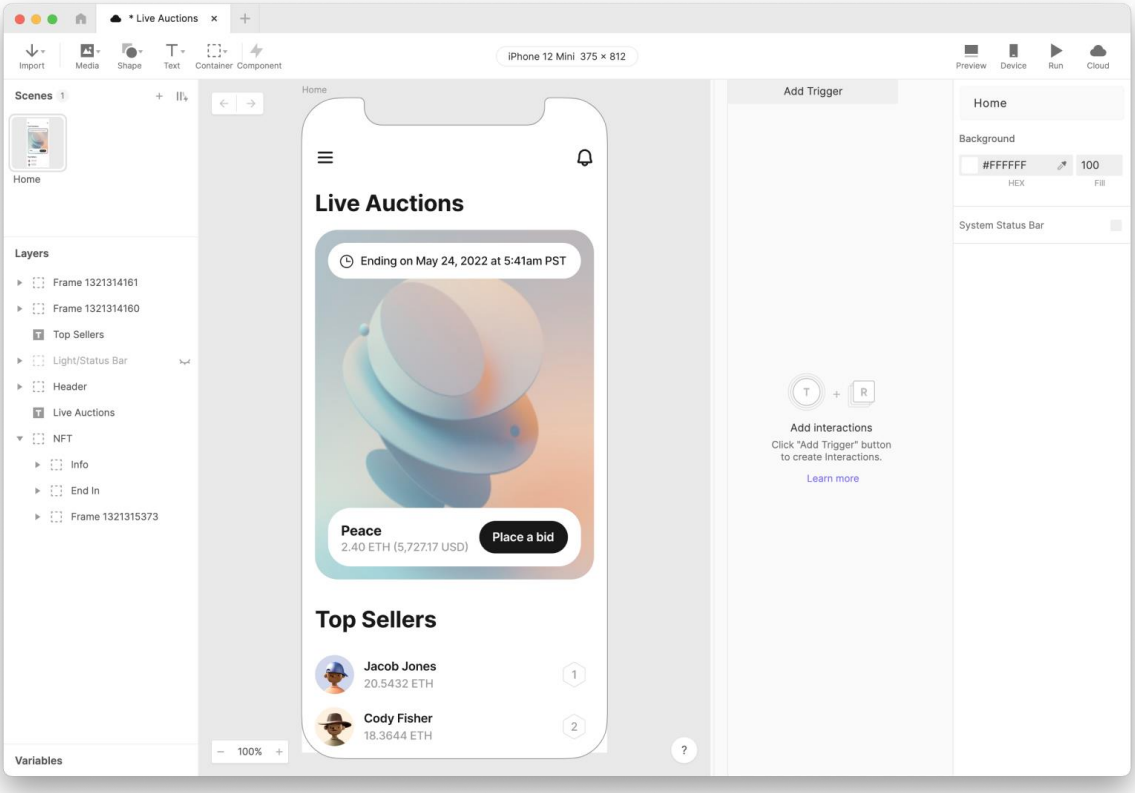

Рисунок 28 – Пользовательский интерфейс ProtoPie

К минусам можно отнести высокие трудозатраты на создание прототипа, которые нецелесообразны в простых проектах, слишком узкая аудитория пользователей.

Sketch – данный инструмент позволяет сосредоточится только на дизайне интерфейсов. Огромное количество сторонних плагинов и большое комьюнити. Sketch был разработан для MacOS. Пользовательский интерфейс инструмента Sketch изображен на рисунке 29.

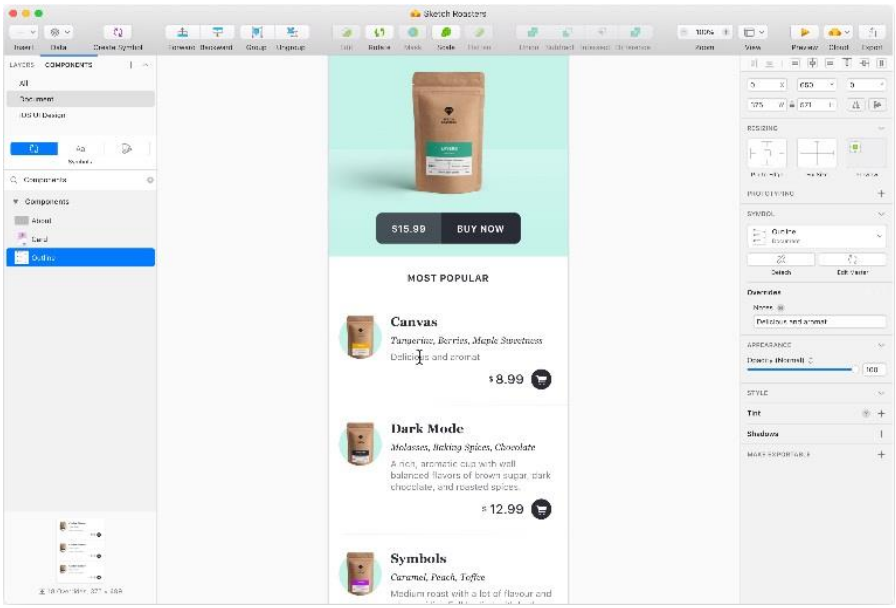

Рисунок 29 – Пользовательский интерфейс Sketch

К минусам можно отнести отсутствие версии для Windows. Помимо этого, при подключении множества сервисов и плагинов система получается огромным и загружается очень долго. Сравнительный анализ инструментов для прототипирование представлена в таблице 7.

Таблица 7 – Сравнение инструментов для прототипирования

Инструмент	Цена	Доступ к прототипу	Уровень сложности	Сообщество

Продолжение таблицы 7

Figma	Есть бесплатная версия навсегда Далее 120$ в год	Онлайн версия, десктопная версия, мобильное приложение	Средний	Много
Adobe XD	Есть бесплатная версия навсегда Далее от 360$ в год	Десктопная (Windows, Mac OS)	Средний	Много
ProtoPie	Есть бесплатная версия на 14 дней Далее 804$ в год	Десктопная (Windows, Mac OS)	Средний	Мало
Sketch	Есть бесплатная версия на 30 дней Далее 99$ в год	Онлайн версия, десктопная (Mac OS)	Средний	Много

Все программные обеспечения имеют свои преимущества и недостатки. Проводя анализ самых популярных программных обеспечений для прототипирования, останавливаемся на инструменте Figma.

Таким образом, после исследования методов и методологий разработки, анализа методов проектирования пользовательского интерфейса, была реализована модель для решения поставленных задач. Подробная модель показан на рисунке 30.

Рисунок 30 – Модель методов и методологий разработки

На основе данной модели будет реализована разработка и интеграция мобильного приложения с КИС «Колледж 24».

Вывод по главе 2

Во второй главе были рассмотрены существующие методы и технологии разработки мобильных приложений, проанализированы способы интеграции информационных систем с мобильными приложениями, а также методы и модели интеграции мобильных приложений.

На основе анализа было определено использование веб-сервисов, а именно REST API в качестве интеграционного шлюза между КИС «Колледж 24» и разрабатываемом мобильным приложением. Спроектированы все методы, которые будут использованы в процессе разработки и интеграции мобильного приложения с КИС «Колледж 24». Исходя из проведенного анализа был сформирован модель интеграции мобильного приложения с КИС, который предоставлен в Приложении А.

Глава 3. Интеграция мобильного приложения к КИС

3.1 Разработка Web-сервиса (API)

Реализуемый программный комплекс (API), согласно модели интеграции, должен стать одним из сервисов в существующем программном обеспечения для приема и передачи запросов со стронных приложений по протоколу HTTP.

Основная архитектура реализуемого решения представляет собой Service API, т.е. серверное приложение, программного приложения «Колледж 24» и является ядром для обмена данных со внешними приложениями. Взаимодействуя со внешними приложениями, серверная часть интеграционного решения будет, получать запросы, обрабатывать их, сохранят нужные данные в базу данных исходя из отправленного запроса и соответственно отправляет ответ при этом реализуя REST API по которому внешние приложения (UI – User Interface) получают данные (см. рисунок 31).

В серверной части программного обеспечения, согласно проектированию «Controller-Service-Repository» распределим на нескольких условных частей:

— интерфейс для взаимодействия с пользовательским приложением (Controller),
— бизнес-логика (Services),
— слой доступа к данным (Repository).

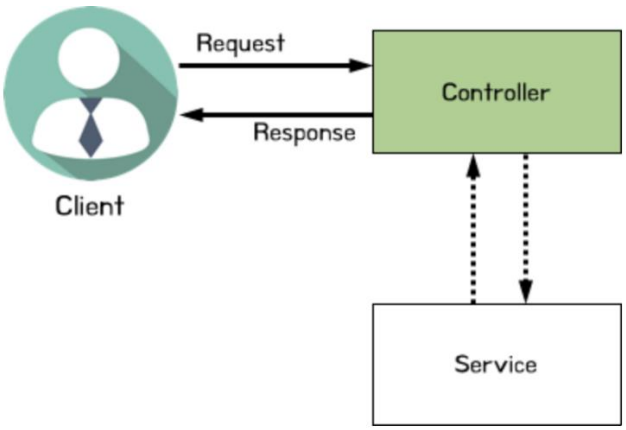

Рисунок 31 - Структура серверного приложения

В серверной части приложения за доступ к данным будет отвечать компонент Repository и этот компонент исключительно содержит функционал по работе с базой данных без бизнес-логики только 3 основные действия как чтение, запись и обновление данных.

Основные функции компонента Repository:

— сохранения логов по каждым запросам,

— получение задач по запросам,

— получение данных по id,

— обновление данных по id,

— удаление данных по id.

Для реализации API, полностью надо описать OpenAPI спецификации при поддержке Swagger и задокументировать каждый endpoint. API модуль будет написан на языке PHP, на котором и написан весь программный продукт «Колледж 24».

Swagger – это фреймворк или набор инструментов, который позволяет автоматически описывать API на основе его кода. Документация, написанная на Swagger, облегчает понимание API для людей и машин. Проектная документация может использоваться для визуализации, тестирование и

генерации клиентского REST API [27]. Документация включает в себя CRUD операции и представлен на рисунке 32.

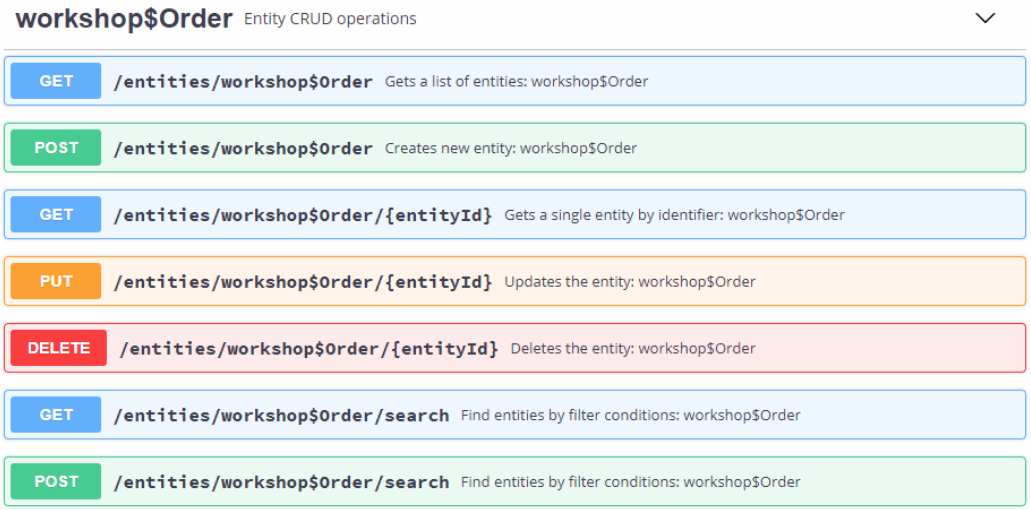

Рисунок 32 – CRUD операции

Объект может представлять собой, например, имя пользователя, название модуля, ссылку или что-то еще. Основных объектов восемь; остальные вложены в них:

— значение openapi — версия OpenAPI, которая используется в проекте;

— info включает в себя вложенные объекты, которые содержат основную информацию об API: название, описание, лицензию, контакты разработчиков и т.д.;

— servers включает в себя ссылки, ведущие к серверам, — базовые пути для доступа извне без учета конечных точек;

— paths описывает конечные точки, или эндпоинты (endpoints) — конец пути к той или иной сущности. Для каждого эндпоинта прописываются запросы GET, POST, DELETE и PUT — операции для взаимодействия с сущностью;

— в components хранятся схемы, которые могут использоваться в разных местах документации. Например, можно создать схему «Пользователь» с полями «Имя», «Адрес» и другими. После описания в components схему можно использовать дальше в коде документации;
— tags хранит метаданные тегов: заголовок, описание и так далее. Помогает более подробно описывать происходящее;
— security описывает методы для обеспечения безопасности, которые можно использовать с API;
— externalDocs содержит ссылки на внешнюю документацию, обычно с дополнительной информацией.

Swagger имеет два способа создания документации. На основе кода и спецификации.

На основе кода, инструмент читает код API, на основе прочитанного генерирует документацию. Данный способ самый легкий и быстрый, разработчику не придется изучать спецификацию, но используется лишь тогда, когда требуется написать документацию в короткие сроки.

На основе спецификации, Swagger, которая называется OpenAPI. Более сложный способ разработки документации, требует знания языка формальных правил. Данный способ более правильный, потому что документация будет понятна конечному разработчику. Для этого способа используется языки JSON и YAML.

Данное решение будем использовать для создания документации и проверки каждой итерации, разрабатываемого интеграционного решения.

Процесс авторизации и получение токена описываем в следующем образе на рисунке 33.

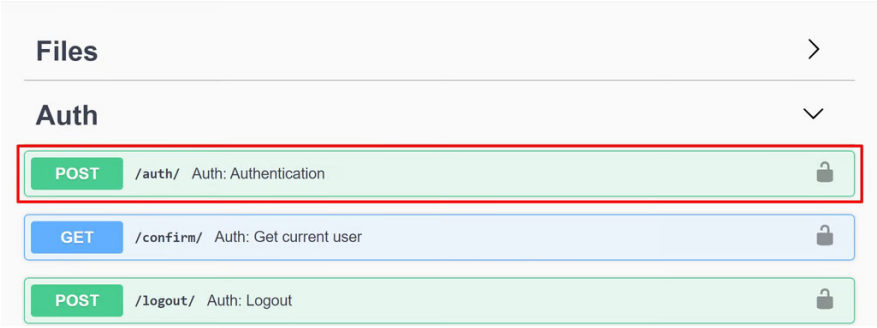

Рисунок 33 - Процесс авторизации

Для получения токена (accessToken) обращаемся к определенному endpoint-у. Вводим пользовательские данные, нажимая Execute отправим запроса для получения токена (см. рисунок 34).

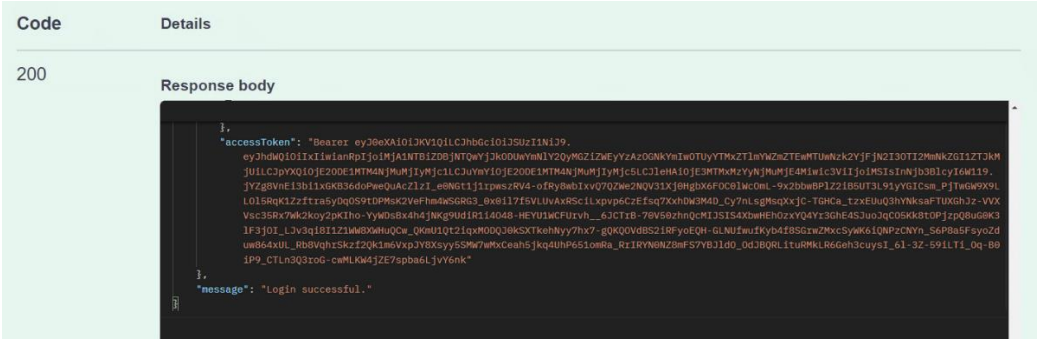

Рисунок 34 - Получения токена (accessToken)

Описываем остальные коды для ориентации и стандарта запросов.

— 401 Unauthorized. Запрос не был успешно обработан, доступа требуется учетные данные (см. рисунок 35).

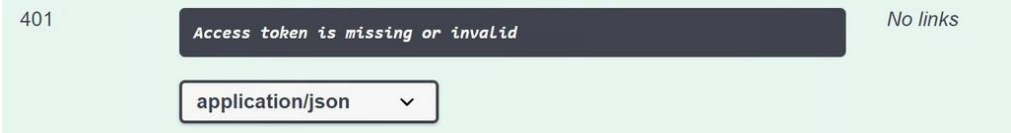

Рисунок 35 - 401 Unauthorized

— 404 Not Found. Запрос был успешно обработан, но сервер не может найти данные по заданному запросу (см. рисунок 36).

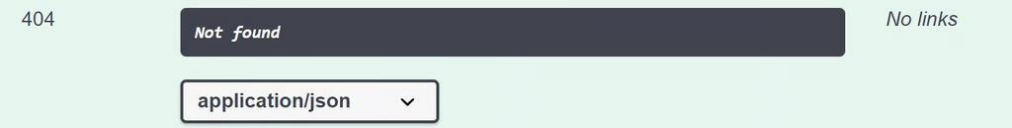

Рисунок 36 - 404 Not Found

— 422 Unprocessable Entity. При запросе неправильно указаны входные данные (см. рисунок 37).

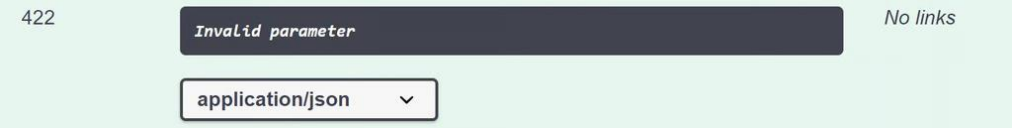

Рисунок 37 – 422 Unprocessable Entity

— 500 Internal Server Error. Внутренняя ошибка сервера, когда клиент отправляет запрос, а тот не может его обработать (см. рисунок 38).

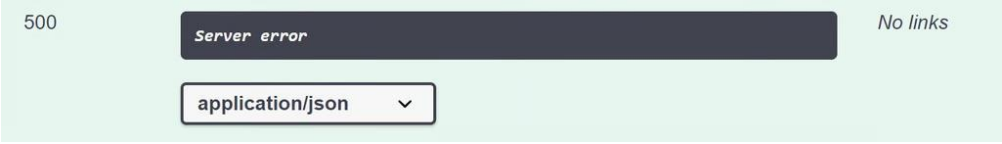

Рисунок 38 - 500 Internal Server Error

Весь интеграционный модуль был разработан и проверен. В ходе разработки интеграционного модуля описание API полностью разработан в фреймворке Swagger.

3.2 Разработка мобильного приложения

Для того чтобы понять какие элементы будут в конечном мобильном приложении, необходимо создать его прототип. Прототипирование — это один из начальных этапов разработки, при котором создаётся предварительный дизайн/эскиз конечного продукта, его структуру со схематичным изображением основных элементов [28].

На данном этапе создается макет, который имитирует взаимодействия пользователя с интерфейсом. Прототип можно рисовать на обычной бумаге или создать в графических редакторах. Главная цель прототипирования это сэкономить время и деньги. Ценность прототипирования в том, что они помогают взглянуть на продукт, его структуру и идею, а также быстро понять концепцию.

Прототипирование решает такие задачи как:

— Лучшая идея. Прототип делается быстро, поэтому можно сделать несколько вариантов и согласовать с заказчиком, а потом выбрать наиболее удачный вариант;

— Выявление ошибок. На этапе прототипирования можно понять недочеты и выявить ошибки конечного продукта, при этом сэкономить меньше времени и деньги чем вносить корректировки на готовом проекте;

— Оценка юзабилити. Оценить удобство для пользователя, и быстро внести изменения в случае выявлений недочетов.

Прототипирование проходит несколько основных этапов:

— Постановка цели. На этом этапе соберутся все участники разработки, среди которых, клиент, дизайнер, маркетолог, копирайтер, программист и все, которые будут участвовать в процессе разработки конечного продукта;

— Исследование. Для качественного прототипирования на данном этапе изучают бизнес-процессы клиента, особенности и целевую аудиторию;

— Гипотеза. Тут важно понять, что дает прототипирование, на какие основные вопросы ответит конечный макет;

— Прототипирование. С учетом всех результатов, сформулированных гипотез создается прототип конечного продукта.

Поэтапно нарисуем начальный дизайн проекта. Экран приветствия приложения будет состоят из 4х шагов (см. рисунок 39). Первый шаг — это синхронизация мобильного приложения с корпоративной информационной системы «Колледж 24», а остальные пояснительная записка.

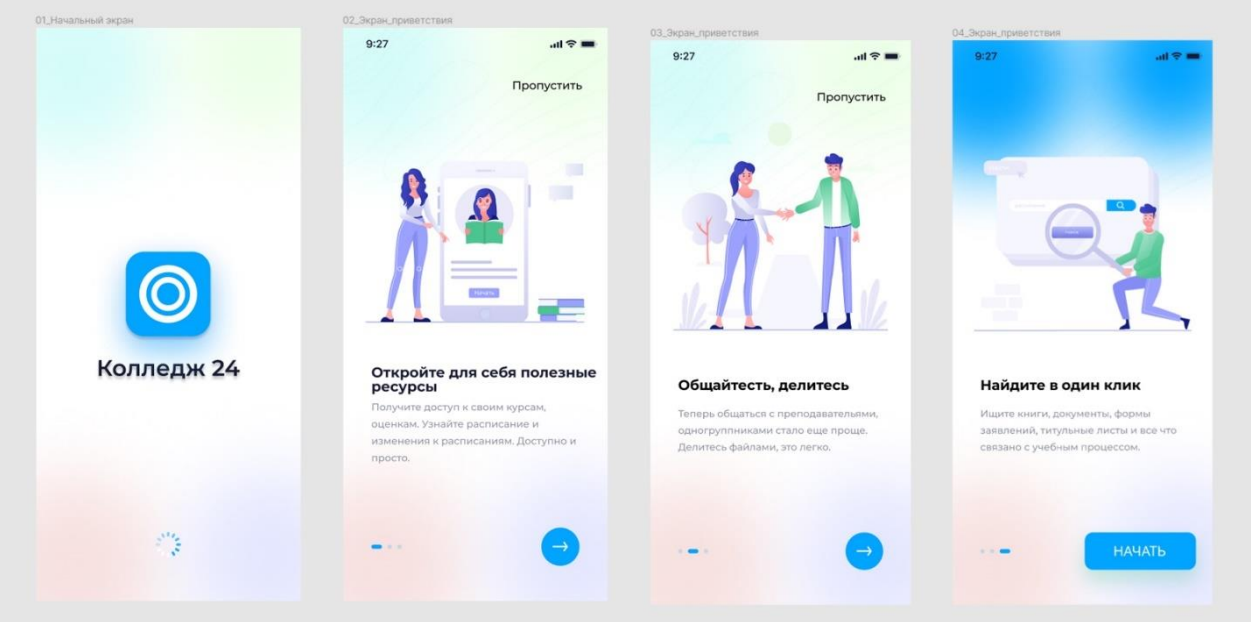

Рисунок 39 - Экран приветствия

Экран авторизации для мобильного приложения состоит из «Экран авторизации», «Получение регистрационных данных», «Восстановление учетных данных», который изображен на рисунке 40.

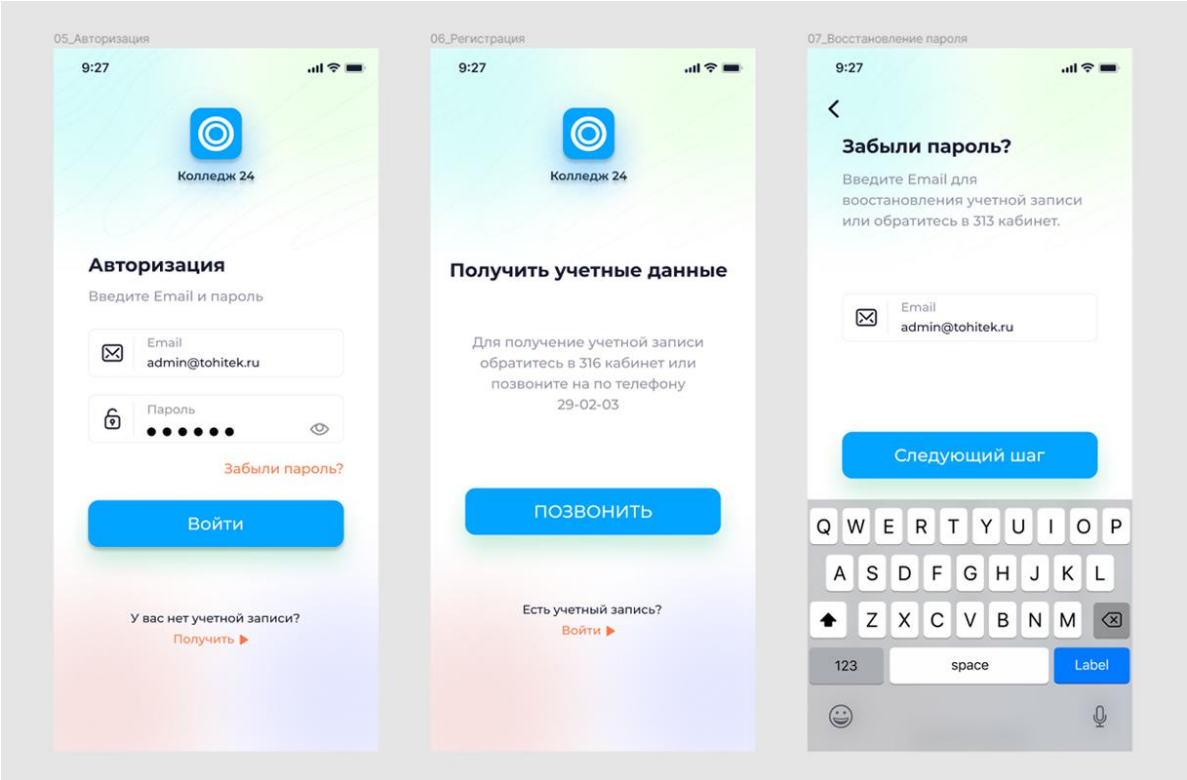

Рисунок 40 - Экран авторизации

Для разработки под фреймворком Flutter нам понадобиться установить IDE. Обычно выбирается среда разработки Android Studio, но, если даже писать код в текстовом редакторе и скомпилировать его в консоли, но это усложняет процесс разработки. Android Studio позволяет создавать приложение на Flutter не только для операционных систем Android и под другие платформы.

Коробочная версия Android Studio не включает в себя фреймворк Flutter, для это после установки среды понадобиться отдельно установить соответствующий плагины Flutter и Dart. На рисунке 41 изображен процесс установки фреймворка Flutter.

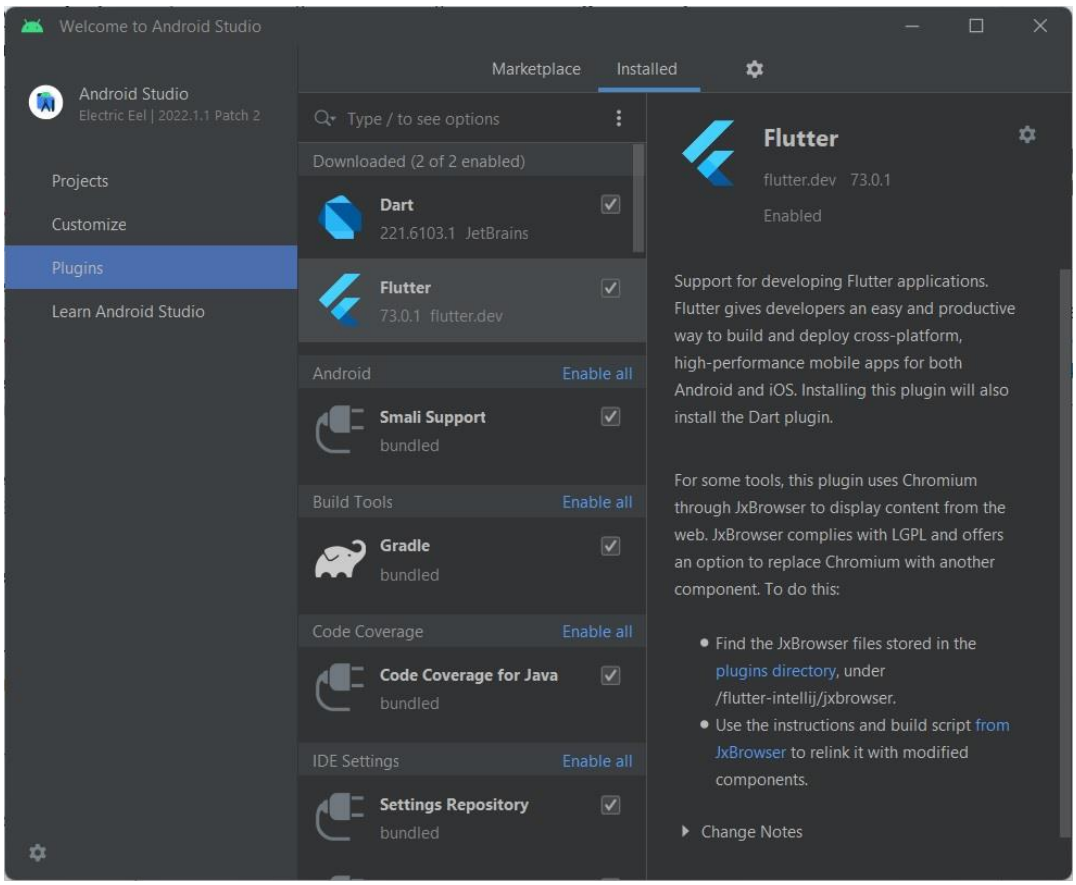

Рисунок 41 – Установка фреймворка Flutter

Первоначальная настройка проекта, и установка зависимостей. После установки плагинов, потребуется установка системы распределенного управления версиями Git и Flutter SDK.

Производим надстройку интерфейса Android Studio и создадим новый проект. Скачиваем последнюю стабильную версию Flutter SDK и в настройках Android Studio указываем путь до конечной папки установленного Flutter SDK.

Процесс переноса дизайна с программы для прототипирования Figma в Android Studio. Создание папок под каждую страницу, и написание стилей уже в Android Studio для каждого элемента. Все файлы стили и изображения будут храниться в специальных папках. На рисунке 42 изображен процесс переноса прототипа на Flutter.

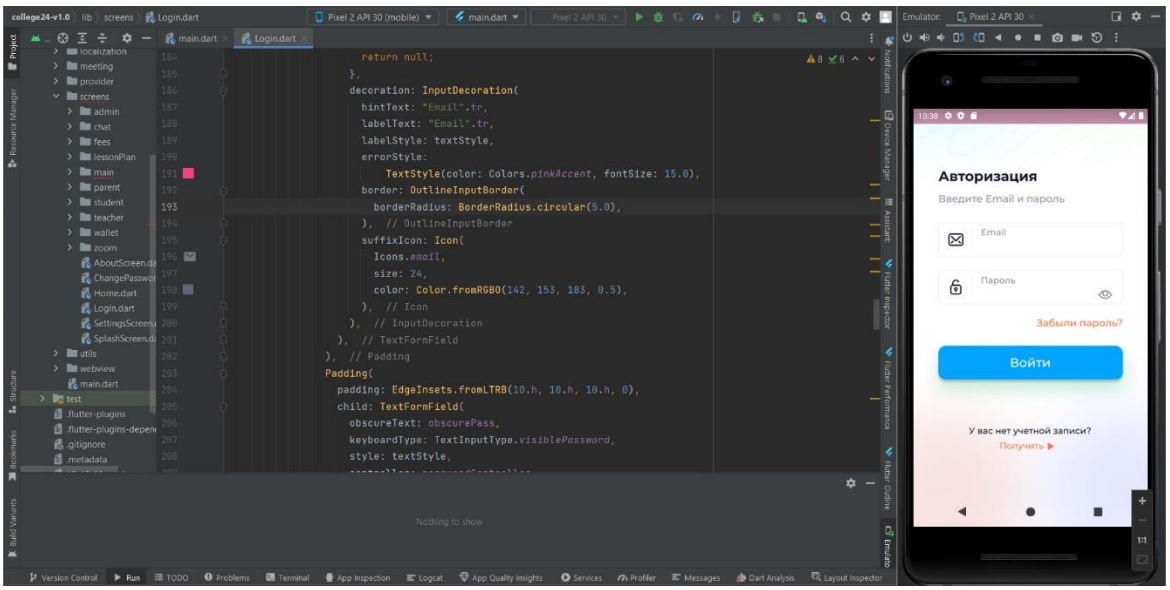

Рисунок 42 - Процесс переноса прототипа

Итоги разработки и переноса дизайна. На данном этапе был создан прототип мобильного приложения. Прототип полностью был перенесен. В ходе переноса были созданы все страницы для каждого пользователя, все стили, иконки, изображения были перенесены уже в готовый код.

3.3 Интеграция мобильного приложения с КИС

В рамках процесса интеграции мобильного приложения с КИС необходимо использовать ранее созданную интерактивную документацию для КИС предприятия.

Мобильное приложение без авторизации после запуска автоматически синхронизируется с КИС «Колледж 24» для получения данных. Без авторизации мобильное приложение синхронизируется и получает такие данные как «Новости/Мероприятия», «Расписание занятий» и сохраняет в базу данных. Процесс синхронизации отображен на рисунке 43. Указанные пункты доступны без авторизации и получения токена.

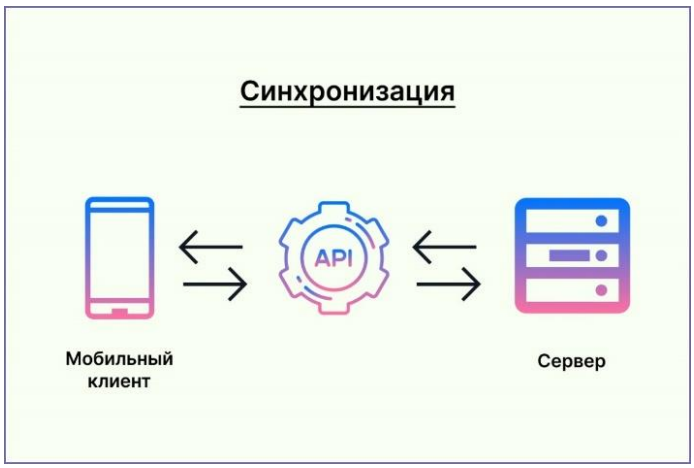

Рисунок 43 - Первоначальная синхронизация

Интеграционное решение имеет пассивно-активный режим. Когда мобильное приложение отправляет запрос на сервер, сервер в свою очередь получает сообщение, ставит их в очередь и, подключаясь к системам-приемникам, доставляет сообщение с ответом обратно (см. рисунок 44). В случае, когда система не может доставить сообщение получателю, ставит их в очередь и повторяет попытку заново, пока сообщение не будет доставлено. Мобильное приложение в свою очередь показывает последний имеющийся в базе данных контент.

Рисунок 44 - Пассивно-активный режим

В рамках интеграции необходимо создать все интеграционные сценарии. Для каждого сценария используем уже созданный шаблон запросов и ответов, создаем функции для мобильного приложения.

После создания сценария синхронизации, приступаем к авторизации и получению токена. В КИС «Колледж 24» был подключен единый протокол авторизации на основе OAuth2. После того как пользователь ведет свои учетные данные, мобильное приложение отправляет запрос авторизации на сервер, а сервер после проверки учетных данных предоставляет разрешение на авторизацию и возвращает токен доступа пользователю. Процесс получение токена описан в рисунке 45.

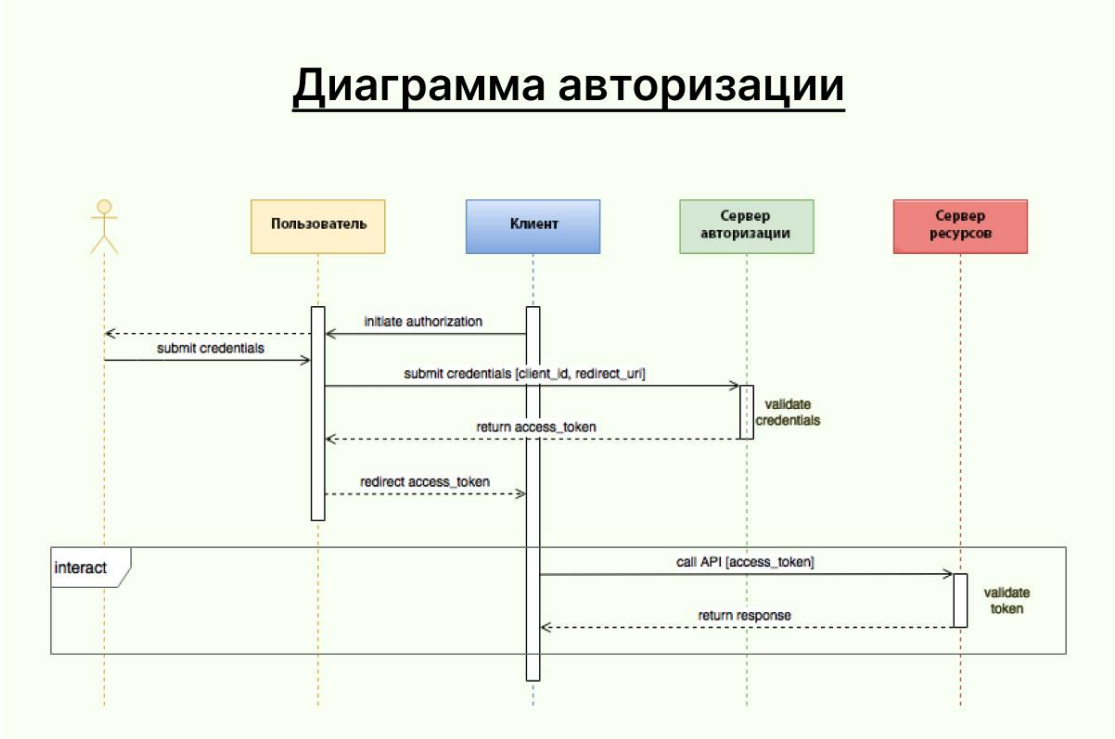

Рисунок 45 - Диаграмма авторизации пользователей

Рассмотрим описание последовательности шагов на диаграммы авторизации:

— Приложение запрашивает у пользователя учетные данные для авторизации;

— Пользователь отправляет запрос, приложение получает разрешение на авторизацию;

— Приложение запрашивает токен у сервера авторизации (API);

— Если предоставленные данные действительны, сервер авторизации (API) создает токен доступа и отправляет приложению. На этом процесс авторизации завершается.

Получение ресурсов работает по предоставлению токена авторизации и сервер предоставляет запрашиваемый ресурс приложению.

На рисунке 46 отражено главное меню после авторизации пользователя в роли студента.

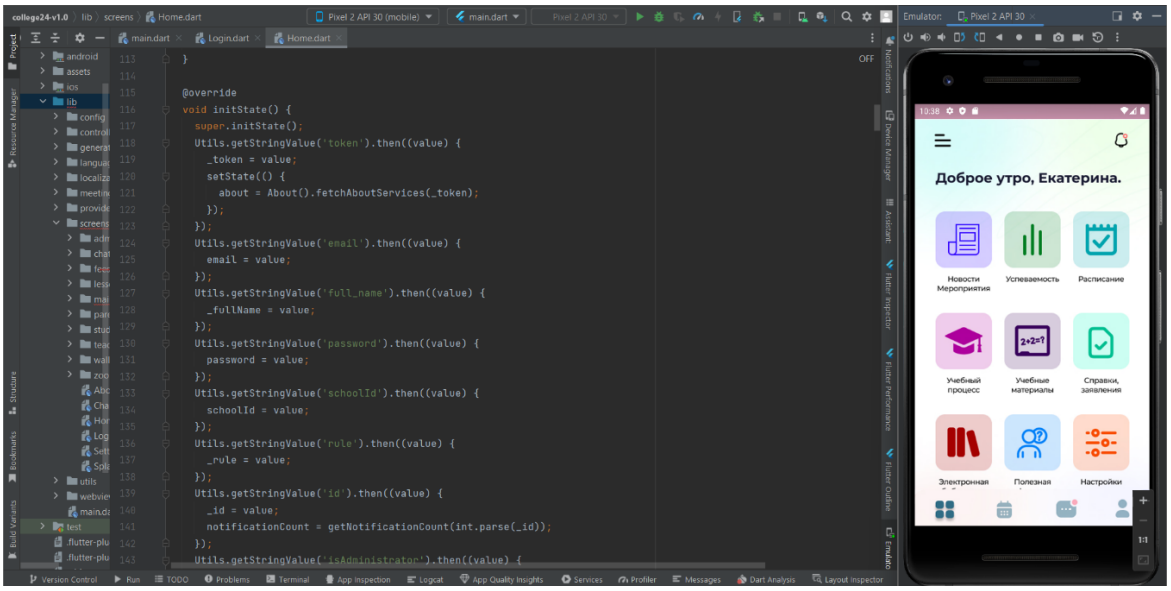

Рисунок 46 - Главное меню авторизованного пользователя

На основе основных функций КИС «Колледж 24» и разработанной документации по RESTful API, связываем функционал мобильного приложения с API информационной системы. Каждый этап интеграции будет тестироваться для получения необходимых данных во время отправки и получения запроса. Интерфейс каждого пользователя в мобильном приложении имеет свои функциональные возможности исходя из его роли.

Вывод по главе 3

На основе выбранных методов разработки и интеграции мобильного приложения с КИС «Колледж 24», был разработан веб-сервис на основе архитектурного стиля взаимодействия компонентов распределенного приложения в сети, RESTful API.

Были проанализированы типы мобильных приложений и для решения поставленных задач был разработан, прототип мобильного приложения в приложении для прототипирования Figma, разработано кроссплатформенное мобильное приложение на основе фреймворка Flutter и языка программирование Dart.

Мобильное приложение было интегрировано с КИС «Колледж 24», на основе архитектурного стиля взаимодействия компонентов распределенного приложения в сети, RESTful API. Каждый этап интеграции протестирован с разными уровнями доступа исходя из роли пользователя.

Глава 4. Анализ эффективности интеграции

4.1 Результат разработки и интеграции мобильного приложения

Тестирование мобильного приложения – это процесс испытания разработанного приложения для проверки соответствия между реальным поведением и ожидаемым, разбор документации и поведение приложения на разных устройствах. После разработки приложения, необходимо проверить поведения приложения таким, каким изначально было спроектировано, его надо протестировать.

Фреймворк Flutter имеет полный набор инструментов для тестирования приложений:

— модульный тест (unit test),

— тест виджетов (widget test),

— интеграционный тест (integration test).

Модульный тест – предназначен для тестирования одной функции, метода или класса. Модульный тест проверяет работоспособность определенной функции, метода или класса.

Виджет тест – для тестирования виджета. Проверка виджета пользовательского интерфейса (UI), одинаковое отображение виджета и его взаимодействия. Данный тест происходит в тестовой среде, которая обеспечивает жизненный цикл виджета и тестируемый виджет имеет возможность получать действия от пользователя отвечая на них.

Интеграционный тест – тестирования приложения частично или в целом. Цель данного тестирования, проверка работоспособности виджетов, сервисов и производительности приложения. Этот этап тестирования происходит на реальном устройстве или эмуляторе.

Хорошее тестирование приложения имеет множество тестов модулей и виджетов, которое отслеживается по покрытию кода, а интеграционные тесты охватывает все варианты использования приложения. Компания Google

разработала ряд компромиссов между различными видами тестирования, которая показана на рисунке 47.

Рисунок 47 – Таблица компромисса

На примере таблицы компромисса, можно сделать вывод, что модульное тестирование дает низкую уверенность в разработанном приложении, но тесты выполняются быстро, а интеграционные тесты самая высокая уверенность при этом выполняются медленнее.

Все файлы тестирования будут находится в папке проекта «test». Необходимо создать 3 файла:

— application_test.dart – общий файл для запуска всех тестов,

— unit.dart – модульное тестирование,

— widget_test.dart – тестирование виджета.

Проводим модульное тестирования и тестирования виджета. После успешного прохождения данных тестов приступим к интеграционному тестированию.

Интеграционные тесты не входят в пакет Flutter. Для этого в файле pubspec.yaml добавить следующие зависимости исходя из виджетов и модулей, которая показана на рисунке 48.

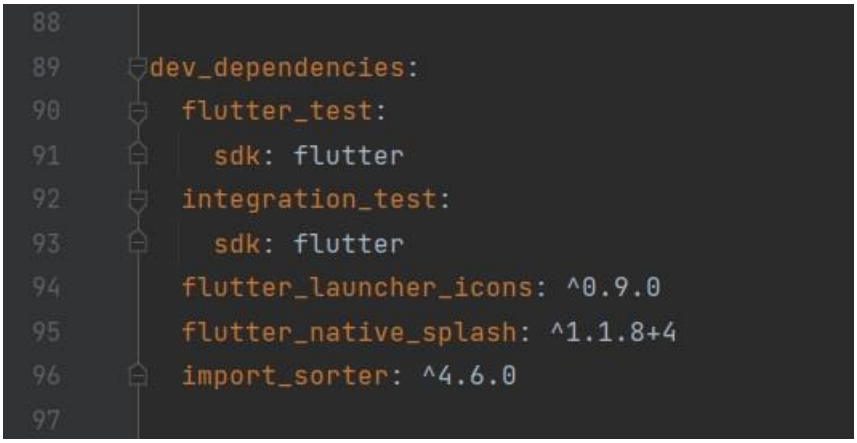

Рисунок 48 – Установка зависимостей

В файле application_test.dart инициализируем службу для запуска тестов на реальном устройстве. Для инициализации добавить следующий код:

```
void main() {
 IntegrationTestWidgetsFlutterBinding.ensureInitialized();
   /// далее код для тестирования
}
```

Для запуска процесса тестирования, в терминале набираем команду:

```
flutter test integration_test/application_test.dart
```

Для полного отчета результата тестирования будем использовать Flutter Dev Tools. Flutter Dev Tools – инструмент для тестирования и отладки Flutter приложений [30]. Данный инструмент имеет множества функций для тестировщика и разработчика, это мониторинг производительности, памяти, сниффер и логирование.

Тестирование мобильного приложения разработанного на фреймворке Flutter на этом этапе заканчивается. Все этапы тестирования пройдены успешно.

Разработанное мобильное приложение не имеет большого количества анимации, быстрых переходов и сложной графики, поэтому не проседает в скорости запуска и производительности после релиза.

4.2 Расчет эффективности интеграции разработанного мобильного приложения

Основным математическим инструментом, для определения и прогнозирования показателей эффективности и надежности, является модель надежности разработанного программного продукта, которая строится для установления зависимостей надежности от заранее запланированных параметров. Эти показатели следует рассматривать в единстве процессов предсказания, измерения и оценивания.

Аналитическое моделирование надежности обычно включают в себя 4 шага:

- определение предположений;
- реализация или выбор модели;
- определение параметров модели с использованием полученных данных;
- применение модели – расчет эффективности показателей надежности модели.

Предсказания – это процесс определения показателей надежности реализуемого проекта. Измерения – это процесс определения показателей надежности на основе анализа о временных промежутках между отказами во время тестирования. Оценивание – это показатель для определения надежности полученных при анализе работы продукта в условиях эксплуатации.

Для решения данных задач можно использовать аналитические и эмпирические модели надежности продукта. Аналитические модели позволяют рассчитать количественные показатели надежности, а

эмпирические модели на структурных и поведенческих особенностях продукта.

Аналитические моделирование надежности делятся на 4 основных этапа:

1. определение стратегии, плана процесса тестирования продукта исходя из обнаруженных ошибок;
2. выбора математической модели или ее создания, на основе принятых стратегий в плане тестирования;
3. выбор параметра моделей на основе полученных данных;
4. расчет количественных показателей надежности с использованием построенной модели.

В качестве анализа эффективности разработанного приложения выберем модель Шумана. Оценка надежности проводиться на основе проведенного тестирования.

Были проведены 3 этапа тестирования. Продолжительность первого этапа составило 9 часов, второго этапа – 16 часов, третьего этапа – 23 часа. На этих этапах было выявлено 5, 2, 3 ошибок. Результаты представлены в таблице 8.

Таблица 8 – результаты тестирования

Этап (k)	Продолжительность (t), ч	Количество ошибок (n)
1	14	5
2	29	2
3	42	3

Сумма продолжительности тестирования составляет t = 48 часов.

Сумма обнаруженных и исправленных ошибок во время тестирования составляет n = 10.

Ошибки, исправленные к началу (i + 1) этапа равно n0 = 0; n1 = 5; n2 = 7; n3 = 10.

Для описания функции надежности по модели Шумана используем формулу (1):

$$R_t(t) = e^{-\lambda_i t} \quad (1)$$

где λ интенсивность появление ошибок. Вычисляется формулой (2):

$$\lambda = (N - n)^* C \quad (2)$$

где N начальное количество ошибок, а C коэффициент пропорциональности и равен (3):

$$C = \frac{\sum_{i=1}^{k} \frac{m_i}{N - n_{i-1}}}{\sum_{i=1}^{k} t_i} \quad (3)$$

Для того, чтобы найти начальное количество ошибок N обратимся к формуле (4):

$$\sum_{i=1}^{k} m_i \frac{\sum_{i=1}^{k} t_i}{\sum_{i=1}^{k} \frac{m_i}{N - n_{i-1}}} = \sum_{i=1}^{k} (N - n_{i-1}) t_i \quad (4)$$

Путем подбора из данного уравнения N = 12.

Значения N нам известны, необходимо найти коэффициент C по формуле (3) и получаем значение C = 0.08.

После получение результата коэффициента C, интенсивность появления ошибок равна:

$$\lambda = (12 – (5 + 2 + 3) * 0.08 = 0.16.$$

Получаем функцию надежности (5):

$$R(t) = e^{-0.16t} \quad (5)$$

По итогам полученных результатов можно сделать вывод, что программный продукт может быть эксплуатирован. В приложение были использованы Lazy Lists и Lazy Grids для крупных UI-элементы, что снижает рендеринг при запуске, а также были соблюдены все требования по разработке пользовательского интерфейса уменьшая количество opacity виджетов и рекомендации по улучшению производительности. Методы такие, как build(),

saveLayer() не применились или частично применились. Длинные списки в приложении появляются по мере того, как пользователь будет скроллить, так конструктор будет рендерить их по появлению. Фреймворк Flutter обеспечивает почти нативную производительность даже в приложениях со сложными визуальными эффектами.

Вывод по главе 4

В этой главе были результаты разработки и интеграции мобильного приложения с информационной системой. Было проведено тестирование разработанного мобильного приложения с 3 способами для получения оптимальных результатов разработки.

Расчет эффективности разработанного продукта был рассчитан по модели Шумана. Результаты расчета полностью соответствуют всем требованиям разработки кроссплатформенных мобильных приложений.

Заключение

Результатом работы является анализ методов и разработка модели интеграции мобильного приложения с КИС.

Для достижения результата были выполнены поставленные задачи:

— Проанализированы архитектура КИС, существующие методы интеграции информационных систем и интеграция с мобильными приложениями;

— Разработана модель интеграции мобильного приложения с КИС.

В процессе выполнения данной ВКР, были проанализированы все необходимые требования, по реализации модели и интеграции мобильного приложения с КИС. Исходя из данных требований была реализована модель интеграции мобильного приложения с КИС «Колледж 24», позволяющая получить напрямую информацию о новостях и мероприятиях, получать отчеты по успеваемости, учебному процессу, учебным материалам, документообороту, запросы по получению справок, формы заявлений, общение между преподавателями и одногруппниками, общие чаты групп. В мобильном приложении каждый пользователь обладает ролью, которая прописана в КИС «Колледж 24», и по которой доступен основной функционал системы.

Исходя из модели интеграции, КИС «Колледж 24» можно интегрировать с другими ИС для обмена данными.

После разработки и интеграции мобильного приложения с информационной системой, было проведено тестирование мобильного приложения, а также анализ эффективности разработанного мобильного приложения. По результатам тестирования и расчета эффективности, можно сделать вывод, то разработанное кроссплатформенное мобильное приложение соответствует всем стандартам по производительности и скорости работы.

Функциональность мобильного приложения может быть расширена путем интеграций с другими ИС, совмещенными с КИС «Колледж 24».

Список используемой литературы

1. Android vs. Apple Market Share: Leading Mobile Operating Systems. [Электронный ресурс]. URL: https://www.bankmycell.com/blog/android-vs-apple-market-share/ (дата обращения: 12.04.2023).
2. Чернышев П. А. Кроссплатформенная и нативная разработка мобильных приложений // Авторские колонки. [Электронный ресурс]. URL: https://rb.ru/opinion/krossplatformennaya-razrabotka/ (дата обращения: 19.08.2022).
3. Майоров Е. Е., Таюрская И. С. Корпоративные информационные системы : учебник. С. Издательство Университета при МПА ЕврАзЭС ; СПб. : 2020 – 128 с.
4. Зараменских Е. П., Кудрявцев Д. В. Арзуманян М. Ю. Архитектура предприятия : учебник. М. Юрайт ; Москва : 2022 – 436 с.
5. Luis Weir, Zdenek Nemec. Enterprise API Management: Design and deliver valuable business APIs : Packt Publishing ; Birmingham, 2019 – 300 с.
6. Терещенко П. В., Астапчук В. А. Архитектура корпоративных информационных систем : учеб. пособие : М. : НГТУ ; Новосибирск, 2019. - 173 с.
7. Klaus-Dieter Gronwald. Integrated Business Information Systems: A Holistic View of the Linked Business Process Chain ERP-SCM-CRM-BI-Big Data : Springer, 2020. 196 с.
8. Данилин А. В. Сервис-ориентированная архитектура (SOA) и архитектура, управляемая моделями (MDA). М. : Известия ; Тула, 2016 - 196 с.
9. Суханов А. Я., Разработка веб-сервисов для научных и прикладных задач : учеб. пособие, М. : АСУ ; Томск, 2022 - 290 с.
10. Alexander S. Gillis, Guide to building an enterprise API strategy // TechTarget. URL:

https://www.techtarget.com/searchapparchitecture/definition/RESTful-API (дата обращения: 26.11.2022).

11. Ньюмен С. Создание микросервисов. М. : Питер ; СПб, 2023 - 624 с.

12. OScill, Информационная системная магистраль: что в себя включает, принципы. Как называется правила обмена данными по шине. // [Электронный ресурс]. URL: https://oscill.ru/2199/informacionnaya-sistemnaya-magistral-chto-v-sebya-vkljuchaet-principy-kak-nazyvaetsya-pravila-obmena-dannymi-po-shine/ (дата обращения: 02.02.2023).

13. Мобильные приложения: особенности разработки для бизнеса и госсектора // [Электронный ресурс]. URL: https://retail-loyalty.org/expert-forum/mobilnye-prilozheniya-osobennosti-razrabotki-dlya-biznesa-i-gossektora/ (дата обращения: 18.02.2023).

14. Черников В. Н. Разработка мобильных приложений на C# для iOS и Android : М. : ДМК Пресс ; Москва, 2020. 189 с.

15. Marco L. Napoli, Beginning Flutter : Wiley. 2019. 528 с.

16. Парамонов И. В. Разработка мобильных приложений для платформы Android : учеб. пособие : М. : ЯрГУ ; Ярославль, 2019 — 88 с.

17. Киселев П. В. Прогрессивные веб-приложения: объединяющая технология для веб- и нативных приложений // Политехнический молодежный журнал. - 2020. - №2. - С. 1-9.

18. Кроссплатформенная разработка мобильных приложений с высокой производительностью и нативным дизайном. // [Электронный ресурс]. URL: https://surf.ru/flutter/ (дата обращения: 22.08.2022).

19. Калиневич Н. Гильванов Р. Г. Разработка кросс-платформенных приложений на языке Dart при помощи фреймворка Flutter // Интеллектуальные технологии на транспорте. - 2021. - №4. - С. 21-27.

20. Сравнение фреймворков для разработки мобильных приложений // [Электронный ресурс]. URL: https://proglib.io/p/top-10-android-freymvorkov-obzor-i-sravnenie-2020-08-05 (дата обращения: 12.05.2022).

21. Топ-10 Android-фреймворков: обзор и сравнение. // [Электронный ресурс]. URL: https://techrocks.ru/2020/08/07/top-10-android-frameworks-overview/ (дата обращения: 13.05.2022).

22. Бороненко Т. А., Кайсина А. В., Пальчикова И. Н., Федоркевич Е. В., Федотова В. С. Основы цифровой грамотности и кибербезопасности : учеб. пособие : М.: СПб ЛГУ ; СПб, 2021 - 431 с.

23. Flutter или Xamarin: что выбрать для кроссплатформенной разработки. // [Электронный ресурс]. URL: https://surf.ru/flutter-ili-xamarin-chto-vybrat-dlya-krossplatformennoj-razrabotki/ (дата обращения: 20.11.2022).

24. Почему мобильное приложение на Flutter — хорошая идея для бизнеса. // [Электронный ресурс]. URL: https://surf.ru/pochiemu-mobilnoie-prilozhieniie-na-flutter-khoroshaia-idieia-dlia-bizniesa/ (дата обращения: 17.03.2022).

25. Виктория Вовк. Модели и методологии разработки ПО. // [Электронный ресурс]. URL: https://gb.ru/posts/methodologies (дата обращения. 19.04.2022).

26. Доминика Деграндис. Визуализируйте работу : М. : МИФ Бизнес ; Москва, 2020 - 230 с.

27. Joshua S. Ponelat and Lukas L. Rosenstock, Designing APIs with Swagger and OpenAPI : Manning. 2022 - 424 с.

28. Ирина Хафизова, Прототипирование // [Электронный ресурс]. URL: https://www.unisender.com/ru/glossary/chto-takoe-prototipirovanie-i-zachem-ono-nuzhno/ (дата обращения: 02.03.2023).

29. Matt Neuburg. Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics : O'Reilly Media ; Sebastopol, 2019 – 680 с.

30. Rex Hartson, Pandra Pyla. The UX Book: Agile UX Design for a Quality User Experience : Morgan Kaufmann ; Burlington, 2018 – 916 с.

Приложения А