ВВЕДЕНИЕ

В зависимости от наших потребностей финансы играют определенную роль в нашей жизни. За большую многовековую историю они стали хорошим инструментом, при помощи которого человек направляет и усиливает свою жизненную энергию, реализует свои планы и идеи. И чтобы это происходило правильно, нужно соблюдать ряд правил, только тогда человек сможет жить в гармонии с собой и окружающим его миром.

Чтобы добиться этой гармонии существует ряд подходов, один из них – ведение бюджета. Бюджет – это своеобразные план использования и формирования денежных средств для реализации необходимых функций и задач.

Ведение бюджета помогает решить ряд проблем: навести порядок в голове; закрыть множество дыр в бюджете, через которые финансы уходят в неизвестном направлении; стать умнее, эффективнее и успешнее; реализовать мечты, и сделать это как можно раньше. В настоящее время вести бюджет – такая же необходимость, как чистить зубы или вести здоровый образ жизни, или развиваться, учиться и стремиться вперед.

Приложение ведение бюджета представляет собой программный комплекс, обеспечивающий хранение, каталогизацию и управление базой затрат в электронном виде, а также доступ к большому числу графиком для анализа бюджета. Системы базы данных выполнена в виде веб-приложения, что позволяет создать единую кодовую базу, а также обеспечивает доступ на любых устройствах, имеющих веб-браузер.

Внедрение кроссплатформенного веб-приложения позволило обеспечить удобный доступ к ведению бюджета, обеспечить возможность анализа бюджета, а также обеспечить удаленный доступ к данным в любое время.

Целью данной работы является изучение и выработка мер по обеспечению высокого уровня эргономичности кроссплатформенного веб-приложения для ведения бюджета. Среди задач можно выделить две основные:

* обеспечение юзабилити пользовательского интерфейса пользователей приложения, сюда входит, в первую очередь, обеспечение удобного интерфейса пользователя для ввода, поиска, просмотра и анализа введенных данных;
* обеспечение доступа к приложению с любых аппаратных и программных платформ при сохранении юзабили пользовательского интерфейса и скорости работы.

Данный вид приложений пользуется большим спросом у большого числа пользователей, что делает обеспечение эргономичности и высокого уровня юзабилити приложения ведения бюджета приоритетной задачей.

**1** АНАЛИЗ ПРИЛОЖЕНИЙ ВЕДЕНИЯ БЮДЖЕТА

## **1.1** Обзор существующих аналогов

Сегодня существует выбор среди приложений для ведения бюджета, обеспечивающих автоматизацию данного процесса. Часть из них каталогизирует затраты, часть выполняет автоматизацию регистрации затрат, некоторые – предоставляют средства для хранения и просмотра общей информации по затратам. Небольшая часть встроена в приложения банков и ведет учет согласно счетам по кредитным картам. Ниже представлены основные программные продукты, обеспечивающие более полный набор функцию по ведению бюджета.

Одним из наиболее приближенных примеров является приложение «Daily Budget» (рисунок 1.1).



Рисунок 1.1 – Главная страница приложения «Daily Budget»

Данный проект – это приложение для ведения бюджета. В нем реализована функциональность введения доходов и расходов. Все доходы и расходы разделяются на группы (рисунок 1.2), что удобно для группировки и представления данных. Это также удобно для отображения графиков ежемесячных затрат для каждой конкретной группы.

Также в приложении предусмотрено указание постоянного ежемесячного дохода, а также постоянных затрат, чтобы не вводить данную информацию ежемесячно вручную. Исходя из ежемесячного дохода и желаемой суммы для сохранения, высчитывается ежедневный бюджет, который доступен пользователю для достижения поставленной цели. Это сделано для того, чтобы наглядно на графике можно было убедиться, вкладывается ли пользователь в положенный ему ежедневный бюджет (рисунок 1.3).

Также, к основным недостаткам приложения относятся:

* разработано только под операционную систему (ОС) iOS;
* нет синхронизации данных для работы с разных устройств;
* все данные хранятся у пользователя в телефоне, что не безопасно;
* ограниченный набор графиков;
* жесткая привязка к работе только с одной валютой.



Рисунок 1.2 – Группы затрат приложения «Daily Budget»



Рисунок 1.3 – График попадания в бюджет приложения «Daily Budget»

Еще один аналог - приложение «ViZi Бюджет» (рисунок 1.4). Это приложения для ведения бюджета предоставляет больший набор функций в сравнении с предыдущим аналогом. Есть возможность синхронизации данных, введения собственных категорий затрат, а также возможность иметь несколько счетов. Приложение имеет бесплатную и платную версии, и представлено для двух мобильных платформ: iOS и Android.



Рисунок 1.4 –Приложение «ViZi Бюджет»

Данное приложение также не лишено недостатков:

* сложный пользовательский интерфейс;
* отсутствие веб-версии приложения;
* слишком большой набор функций для новичка.

## **1.2** Аналитический обзор

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами. Книга Сэмма Руби «Гибкая разработка веб-приложений» [1] содержит информацию о том, как создать простое веб-приложение с нуля. В книгу включены описание простейшего рабочего процесса (с использованием текстового редактора и системы контроля версий Git), основы технологий клиентской стороны (HTML, CSS, jQuery, JavaScript), основы серверных технологий (HTTP, базы данных), основы облачного развертывания и несколько примеров правильной практики написания кода (функции, MVC). С ее помощью можно изучить фундаментальные основы языка, научиться программировать, используя объекты и массивы, а также ментальные модели, которые соответствуют этому типу разработки программного обеспечения (ПО).

Мобильное приложение – программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах [2]. Мобильные приложения ушли далеко вперед в связи с развитием сотовой связи и беспроводных технологий (3G, 4G или LTE, Wi-Fi, WiMax).

Многие приложения могут быть предустановлены на устройство в процессе производства, загружены пользователем с помощью различных платформ для распространения ПО или существовать в формате веб-приложений.

Основные операционные системы, на базе которых строятся мобильные приложения – iOS, Android и WindowsMobile.

Существуют различные каналы распространения приложений: специализированный онлайн магазин – AppStore, AndroidMarket, MicrosoftStore, самостоятельный поиск или скачивание в Интернете.

## **1.3** Архитектура клиент-сервер

Важным вопросом в разработке веб-приложений является их архитектура. Наиболее эффективную работу приложений обеспечивает архитектура «клиент-сервер» [3].

Особенностью архитектуры является то, что само веб-приложение находится и выполняется на сервере, клиент при этом получает только результаты работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата. Вся передача запросов и результатов их обработки происходит через Интернет.

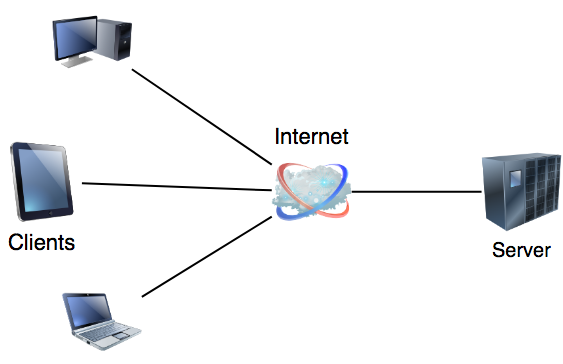


Рисунок 1.5 - Архитектура клиент-сервер

Приемом данных от клиента и их передачей на сервер, а также отображением результатов запросов обычно занимается специальное приложение – браузер (Microsoft Edge, Firefox, Safari, Google Chrome). Одной из функций браузера является отображение данных, полученных из Интернета, в виде страницы, описанной на языке HTML, следовательно, результат, передаваемый сервером клиенту, должен быть представлен на этом языке.

На стороне сервера веб-приложение выполняется специальным программным обеспечением (веб-сервером). Основные задачи, которые он решает – принимает запросы клиентов, обрабатывает их, формирует ответ в виде страницы, описанной на языке HTML, и передает его клиенту.

В процессе обработки запроса пользователя веб-приложение компонует ответ на основе исполнения программного кода, работающего на стороне сервера, веб-формы, страницы HTML, другого содержимого, включая графические файлы.

В результате, как уже было сказано, формируется HTML-страница, которая и отправляется клиенту. Получается, что результат работы веб-приложения идентичен результату запроса к традиционному веб-сайту, однако, в отличие от него, веб-приложение генерирует HTML-код в зависимости от запроса пользователя, а не просто передает его клиенту в том виде, в котором этот код хранится в файле на стороне сервера. То есть веб-приложение динамически формирует ответ с помощью исполняемого кода – так называемой исполняемой части. За счет наличия исполняемой части, веб-приложения способны выполнять практически те же операции, что и обычные Windows-приложения, с тем лишь ограничением, что код исполняется на сервере, в качестве интерфейса системы выступает браузер, а в качестве среды, посредством которой происходит обмен данными, – Интернет.

К наиболее типичным операциям веб-приложения относятся:

* прием данных от пользователя и сохранение их на сервере;
* аутентификация пользователя и отображение интерфейса системы, соответствующего данному пользователю;
* отображение постоянно изменяющейся информации;
* выполнение различных действий по запросу пользователя: извлечение данных из базы данных (БД), добавление, удаление, изменение данных в БД;
* проведение сложных вычислений.

К основным достоинствам архитектуры «клиент-сервер» можно отнести следующие:

* все данные хранятся на сервере, который защищен лучше большинства клиентов;
* использовать ресурсы сервера могут клиенты с разными аппаратными платформами и операционными системами;
* возможность распределить функции вычислительной системы между несколькими компьютерами в сети, что позволяет упростить обслуживание системы;
* на сервере проще организовать контроль прав и полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами;
* позволяет объединить различные клиенты.

Среди недостатков можно выделить необходимость квалифицированного администратора данной системы. В случае использования централизованной системы, неработоспособность основного сервера может сделать неработоспособным все приложение. Немаловажным фактором является высокая стоимость оборудования.

## **1.4** REST-сервис

REST – это набор архитектурных принципов и стиль проектирования приложений, ориентированный на создание сетевых систем, в основе которых лежат механизмы для описания и обращения к ресурсам [4].

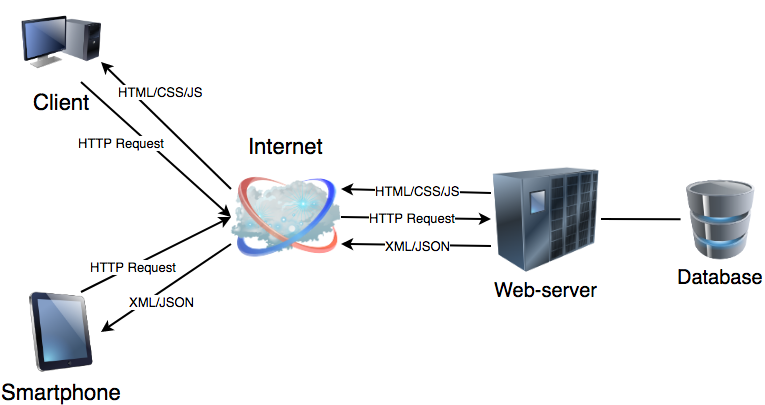


Рисунок 1.6 - REST-сервис

Примером такой системы служит World Wide Web – распределенная система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключенных к сети Интернет [5].

REST определяет строгое разделение ответственности между компонентами клиент-серверной системы, облегчающее реализацию необходимых актеров. Еще одна цель REST – упрощение семантики взаимодействия компонентов сетевых систем для улучшения масштабируемости и повышения производительности. В REST заложен принцип автономности запросов, который означает, что запросы, обрабатываемые сервером или клиентом, обязательно должны включать всю информацию, необходимую для их понимания.

Для обмена данными стандартных медиа-типов в системах REST используется минимальное количество запросов. REST-системы используют URI (универсальные идентификаторы ресурсов) для поиска и получения доступа к представлениям необходимых ресурсов. В течение последних нескольких лет разработчики создавали REST-сервисы для своих приложений, используя самые разнообразные технологии. Также архитектура REST отличается своей простотой, требуя от приложений обеспечить только возможность приема сообщений с HTTP- заголовками.

## **1.5** Apache Cordova

Apache Cordova — это платформа разработки мобильных приложений с открытым исходным кодом. Она позволяет использовать стандартные веб-технологии, такие как HTML5, CSS3 и JavaScript для кроссплатформенной разработки, избегая родного языка разработки для каждой из мобильных платформ. Приложения выполняются внутри обертки нацеленной на каждую платформу и полагаются на стандартные API (программный интерфейс приложения) для доступа к датчикам устройства, данным и состоянию сети [6].

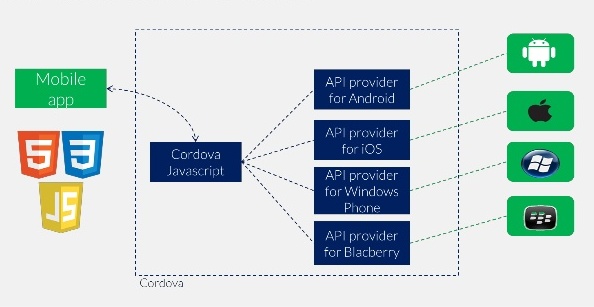


Рисунок 1.7 - Программный интерфейс приложения Cordova [7]

Основные причины использования Apache Cordova:

* нужно расширить приложение на более чем одну платформу, без необходимости повторно реализовать его для каждого языка разработки платформ и набора инструментов;
* нужно развернуть веб-приложение, которое упаковано для распространения в различных магазинах приложений;
* необходимо смешивание компонентов собственного приложения с WebView (специальный браузер окно).

Само приложение реализовано как веб-страницы, по умолчанию локальный файл под названием index.html, который ссылается на любой CSS, JavaScript, изображения, файлы мультимедиа или другие ресурсы необходимы для его запуска. Приложение выполняет как WebView в пределах оболочки приложения, которую вы распространяете в магазины приложений.

Таким образом, с помощью данной платформы есть возможность писать слабонагруженные приложения, который будут работать на любой мобильной платформе. При этом затраты на разработку будут минимальны, поскольку код будет написан один раз, а запускаться будет как в браузере, так и на любом мобильном устройстве.

**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты разрабатываемой системы и выработав список требований необходимых для разработки системы, приложение разбивается на функциональные блоки (модули). Это необходимо для обеспечения гибкой архитектуры. Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В разрабатываемом веб-приложении можно выделить следующие блоки:

* блок пользовательского интерфейса;
* блок ядра клиентского приложения;
* блок для работы с клиентской базой данных;
* блок работы с клиентским приложением;
* блок регистрации;
* блок веб-сервиса;
* блок работы с реляционной базой данных;
* блок ядра серверного приложения.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.216 C1.

Каждый модуль выполняет свою задачу. Чтобы система работала, каждый модуль взаимодействует с другими модулями путем обмена данными, используя различные форматы и протоколы.

Рассмотрим функциональные блоки веб-приложения.

*Блок пользовательского интерфейса* является клиентской частью веб-приложения. Данный блок представляет собой совокупность средств, при помощи которых пользователь взаимодействует с приложением через браузер. Для построения интерфейса используется технологии HTML, CSS, TypeScript.

TypeScript — язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющий возможности JavaScript. Строгость и большая функциональность языка позволяет разработать библиотеку компонент, с помощью которой будут строиться основные компоненты клиентского интерфейса [8].

*Блок для работы с клиентской базой данных* предоставляет доступ к локальному хранилищу браузера, где хранятся авторизационные данные пользователя, а также данные, которые по тем или иным причинам не были отправлены на сервер.

Интернет-хранилище – это программные методы и протоколы веб-приложения, используемые для хранения данных в веб-браузере. Интернет-хранилище представляет собой постоянное хранилище данных, похожее на куки, но со значительно расширенной емкостью и без хранения информации в заголовке запроса HTTP. Существуют два основных типа веб-хранилища: локальное хранилище и сессионное хранилище, ведущие себя аналогично постоянным и сессионным кукам соответственно.

Таким образом, данный блок представляет собой временное хранилище данных пользователя до ближайшего сеанса синхронизации с сервером.

*Блок ядра клиентского приложения* представляет собой центр управления всеми частями клиентской части приложения. Данный блок знает обо всех блоках на клиентской стороне, организует совместную работу всех блоков, а также организует работу с блоком для работы с клиентским приложением.

*Блок работы с клиентским приложением* представляет собой центр для связи клиентской части с серверной. Через него проходят все операции синхронизации и обмена данными.

*Блок регистрации* служит для регистрации новых пользователей приложения. Поскольку регистрация пользователей приложения содержит логику отличную от основной логики работы приложения, имеет смысл вынести его реализацию в отдельный блок.

*Блок веб-сервиса* представляет встроенное приложение с использованием фреймворка ExpressJS, который предназначен для удобного построения API веб-сервисов.

*Блок работы с реляционной базой данных* включает механизмы для взаимодействия с данными, используемыми приложением. Для ее реализации была выбрана база данных SQLite. SQLite – компактная встраиваемая реляционная база данных, запросы к которой можно осуществлять при помощи языка запросов SQL. База данных не поддерживает все особенности SQL и уступает в функциональности другим развитым СУБД, но вполне подходит для хранения и извлечения информации.

SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются функции (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счет того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счет создания файла журнала [9].

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

Благодаря архитектуре движка возможно использовать SQLite как на встраиваемых системах, так и на выделенных машинах с гигабайтными массивами данных.

*Блок ядра серверного приложения* представляет собой центр управления всеми частями серверной части приложения. Данный блок знает обо всех блоках на серверной стороне, организует совместную работу всех блоков.

Блок сервера работает на платформе NodeJS. NodeJS является платформой для создания современных веб-сервисов. Он позволяет писать асинхронный, неблокирующий код, используя стиль и шаблоны NodeJS. Позволяет использовать возможности баз данных, таких как Elasticsearch и Redis. Работать со многими протоколами, создавать веб-службы RESTful, клиенты и серверы сокетов TCP. В платформу встроена возможность проверять функциональность кода с помощью библиотеки Mocha и управлять его жизненным циклом с помощью менеджера пакетов NPM (Node Packet Manager). Используя ряд практических областей программирования, возможно использовать новейшие доступные функции ECMAScript, классы, и популярные модули, а также создавать богатые инструменты командной строки и веб-интерфейс с использованием современных технологий веб-разработки [10].