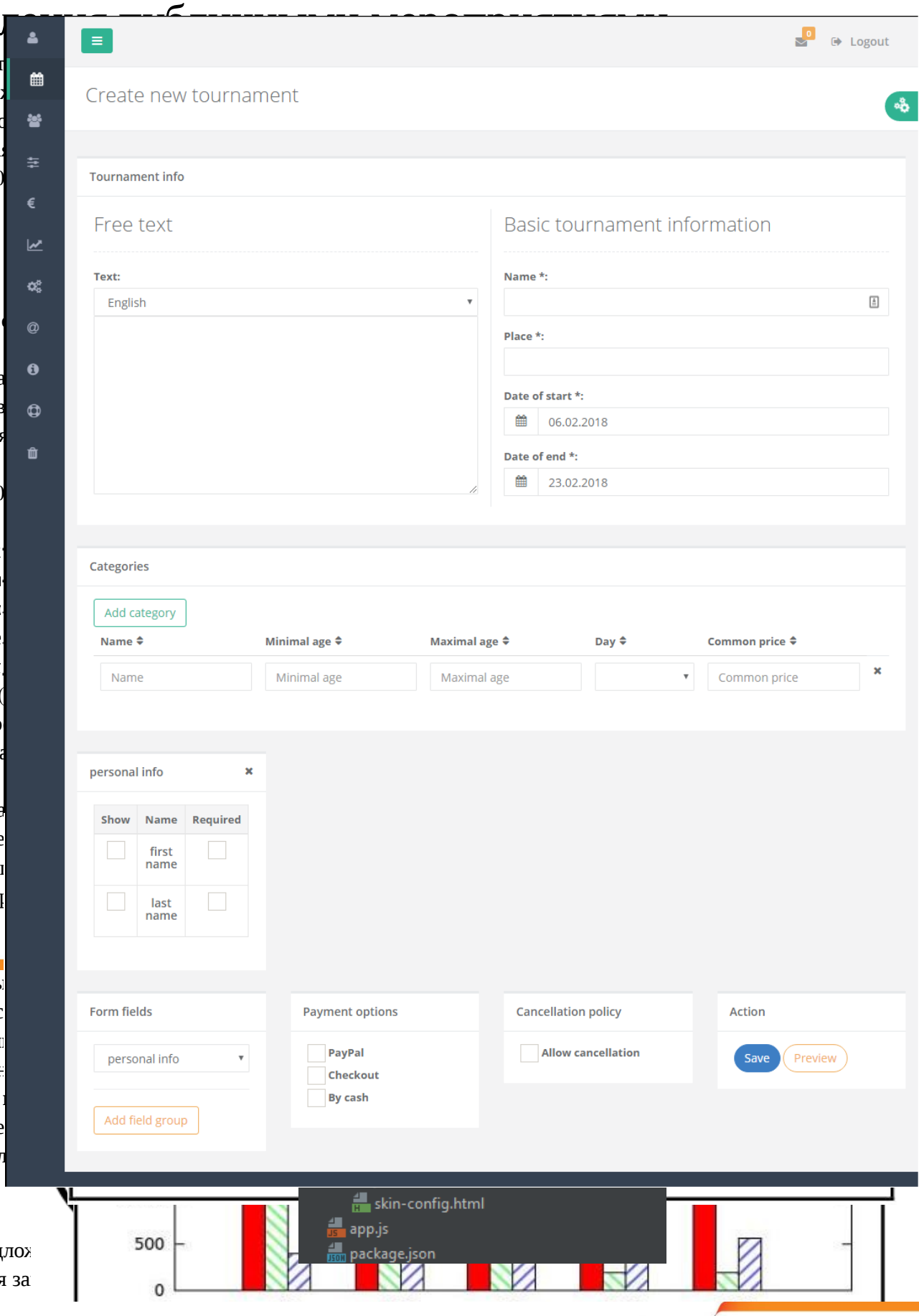
Кирилл Казанцев

Разработка веб-приложения для управ

Университет
Бакалавр ин
Информацио
Бакалаврска
15 апреля 20
года
Аннотация
Автор (ы)
Название
Количество
Дата
Кирилл Каза
Разработка в
мероприятия
49 страниц
15 апреля 20
Степень
Бакалавр ин
Дипломная
Информацио
Вариант спе
Разработка
Инструктор(
Киммо Саур
Эта диссерта
основана на
с использова
Был проведе
и Express бы
проведено с
После про-

танцевальны
. Прототип с
привлекател
осуществлен
платежей и,
Приложение
который был

гибкость.
Проект предло
Диссертация за



Ключевые слова

Node.js , MongoDB, Экспресс, JavaScript

Содержание

Сокращения

АЯКС
Асинхронный JavaScript и XML
CSS
Каскадные таблицы стилей
СУБД
Система управления базами данных
HTML
Язык разметки гипертекста
HTTP
Протокол передачи гипертекста
JS
JavaScript
ИДЕ
Интегрированная среда разработки
IIS
Информационные услуги в Интернете
РСУБД
Системы управления реляционными базами данных
СПА
Одностраничное приложение
SQL

Язык структурированных Запросов
URL
Единый Локатор Ресурсов
2 (45)

1

Введение

Всего несколько десятилетий назад общество было охвачено идеей эры передовых технологии. Сегодня это уже не идея, это реальность. В настоящее время Интернет - это самая быстроразвивающаяся среда обмена информацией в истории человечества.
Трудно назвать какую-либо сферу деятельности, которая не имела бы своей полной и всесторонней размышления в Интернете.
Однако все это было бы трудно представить без веб-приложений. В 1995 году Netscape представил клиентский скриптовый язык под названием JavaScript и, начиная с этого мо-

таким образом, историю веб-приложений можно проследить. Тем не менее, само понятие “веб-

приложение” было введено только в 1999 году.

Как правило, веб-приложения эволюционировали от статических веб-страниц до динамических веб-сайтов, которые обеспечивают индивидуальный пользовательский интерфейс на основе на информации, известной или предоставленной пользователем. Веб-приложения используются для большого разнообразия цели, например, системы назначений, службы мгновенного обмена сообщениями, коммерция, развлечения и даже демократическое принятие решений в некоторых странах.
Этот тезис посвящен внедрению веб-приложения для управления танцевальные турниры. Приложение позволяет пользователям создавать индивидуальные турниры с помощью привлекательный и удобный интерфейс веб-браузера. В то же время приложение дает возможность легко регистрироваться на турниры и производить платежи. Это также может быть используется для управления всеми турнирами и зарегистрированными участниками. Данные и функции-доступность, доступная пользователям, основана на различных правах доступа. Все данные, относящиеся к приложение хранится в базе данных, которая расположена на стороне сервера. Сервер получает и обрабатывает все данные, которые отправляются из веб-браузера, и отправляет обратно все запрошенные данные.
3 (45)

2

Теоретические основы

2.1
Веб-приложение
Веб-приложение - это клиент-серверное приложение, в котором браузер выступает в качестве клиента и веб-сервер как сервер. Логика веб-приложения распределяется между клиентом и сервер, хранение данных осуществляется в основном на сервере. Обмен данными осуществляется по сети-работайте с использованием протокола передачи гипертекста (HTTP). Одно из преимуществ такого подхода является ли тот факт, что пользователи не зависят от конкретной операционной системы или аппаратной конфигурации - рацион. Следовательно, веб-приложения являются кроссплатформенными сервисами.
2.2
Интерфейс или на стороне клиента
Разработка интерфейса - это процесс создания общедоступной части веб-сайта, которая находится в

прямой контакт с пользователем. Кто-то может сказать, что интерфейсная разработка - это всего лишь создание красивого и привлекательного веб-сайта [1]. Это, безусловно, очень важная часть процесс, но существует множество различных технологий, которые подпадают под сферу клиент-побочная разработка [1]. Основными инструментами, которые используются в этом процессе, являются: Гипертекстовая разметка
язык (HTML), Каскадные таблицы стилей (CSS) и JavaScript (JS).
HTML - это основа веб-страницы. Это язык разметки, который определяет общую структура страницы. [1] Использование элемента или тегов, таких как список, таблица, заголовок, разработчик
может маркировать различные части содержимого [2]. Язык HTML интерпретируется браузерами а затем форматированный текст, полученный в результате интерпретации, отображается на компьютере экран или мобильное устройство.
CSS позволяет веб-дизайнерам и разработчикам стилизовать и настроить каждый компонент, который является
определено в HTML [3, 455]. CSS был в первую очередь предназначен для принудительного разделения содержимое веб-страницы в стиле веб-страницы, включая такие функции, как шрифты, цвета и расположение
вон. Такое разделение позволяет нескольким HTML-документам совместно использовать стиль, указанный в
отдельный файл .css. [3]
JavaScript является основой для всех динамических и интерактивных функций на веб-страница. Это легкий и интерпретируемый язык программирования [4]. Его можно использовать, например, чтобы проверить достоверность введенных пользователем данных или изменить

структура веб-страницы на основе захваченных событий, инициированных пользователем, таких как мышь
щелчки. Более того, одной из наиболее полезных функций JS является асинхронный JavaScript и XML (AJAX), этот метод может использоваться для отправки на сервер и получения с него всех требуемые данные без обновления веб-страницы.

2.3 Серверная часть

Внутренняя разработка относится к реализации серверной части, которая в первую очередь предназначена для
ссылки на логику веб-приложения или, другими словами, на то, как работает приложение. Это про-процесс создания ядра веб-приложения, разработка платформы для приложения
и наполнение его всем необходимым функционалом. Серверная часть управляет данными, которые повторно
получен из интерфейса и возвращает результат обратно в понятной форме
на стороне клиента. Серверная часть обычно состоит из трех основных частей: программного обеспечения веб-сервера,
логика приложения и база данных.

2.3.1 Программное обеспечение веб-сервера

Программное обеспечение веб-сервера - это программа, которая работает на аппаратном обеспечении и обслуживает данные для
клиенты, которые обычно представлены в виде браузеров. Программное обеспечение веб-сервера состоит из нескольких
части, но ядром является HTTP-сервер [5]. Это программное обеспечение, которое знает, что такое Униформа
Локатор ресурсов (URL) и понимает протокол HTTP [5]. Основной принцип
связь клиент-сервер с использованием HTTP может быть продемонстрирована на рисунке ниже.
Рисунок 1. HTTP-связь клиент-сервер
Скопировано из MDN [5]

5 (45)
Как показано на рисунке 1, браузер отправляет запрос, затем он попадает на выделенный жесткий ware, а затем HTTP-сервер отправляет ответ, содержащий информацию, которая была повторно подвергнут сомнению. При использовании протокола HTTP всегда соблюдается определенный набор правил [5]. Во- первых,
в конце концов, сервер может ответить только на запрос, который был отправлен клиентом. Это не может отправляйте запросы в браузер. Во-вторых, сервер должен отправлять ответ на каждый входящий поступающий запрос, по крайней мере, ответ, содержащий сообщение об ошибке. [5] Поверх, клиент получит сообщение о тайм-ауте, указывающее, что сервер не ответил
в течение указанного времени. Наконец, каждый HTTP-запрос должен содержать URL-адрес, который указывает конкретный сервер и путь, на который должен быть отправлен этот запрос [5].
Существует множество различных вариантов на выбор при выборе веб-сервера. Среди к ним относятся: Internet Information Services (IIS) от Microsoft, nginx от NGINX, Apache HTTP-сервер, веб-сервер Google и т.д. Рыночная доля каждого веб-сервера по состоянию на Сентябрь 2016 года, можно увидеть на следующем рисунке.
Рисунок 2. Разработчики веб-серверов: рыночная доля активных сайтов
Скопировано из Netcraft [6]
Как показано на рисунке 2, самым популярным веб-сервером на данный момент является Apache с более 46% рыночной доли. Следующим популярным сервером является nginx, на который приходится около 19%.
Выбор веб-сервера основан на различных аспектах, таких как операционная система- тем, уровень безопасности и мощность производительности.
6 (45)
2.3.2 Логика приложения
Логика приложения также называется бизнес-логикой сервера. Это включает в себя все операции при обработке запрошенных и отправленных данных, сохранении данных в базе данных, внесении решения о том, какие данные требуются, а затем запрос необходимых данных из базы данных- основание. Это позволяет использовать меры безопасности, такие как создание механизмов аутентификации
для идентификации пользователя, который запрашивает или отправляет данные с сервера и на сервер. Авторизация-
это еще один пример меры безопасности, которая должна быть реализована в логике приложения. Он используется для проверки, обладает ли данный конкретный пользователь необходимыми правами для выполнения желаемого
операция.
Бизнес-логика создается программно с использованием различных языков программирования, таких как Java, PHP, Ruby, Python, .Net и JS в случае использования Node.js серверное программное обеспечение.
В настоящее время это почти всегда делается с помощью серверных фреймворков. Веб-приложение- катионные фреймворки облегчают разработку, охватывая и предоставляя некоторые основные обычно выполняемые функции, такие как управление сеансами, аутентификация
механизмы, форматирование выходных данных, взаимодействие с базой данных и т.д. [8]. Пример серверным фреймворком может быть CakePHP, Ruby on Rails, Express, Spring, Django и
и т.д.
Таким образом, исходя из вышесказанного, логика приложения является одной из наиболее важных частей бэк-
завершите разработку, которая предписывает способы отображения, создания данных, сохраненный и управляемый. Неоспоримая важность и общее положение бизнеса- логику несс при разработке приложения можно продемонстрировать на рисунке ниже.
7 (45)
Рисунок 3. Типичная структура приложения
Скопировано из MSDN [7]

Основываясь на рисунке 3, можно видеть, что бизнес-логика занимает одну из наиболее важных частей в структура приложения. Внедрение всех необходимых правил, ограничений и процессов последовательности достигается путем кодирования бизнес-правил реального мира в код, не-понятный машинами. Разработчики создают специальные рабочие процессы как часть приложения логика взаимодействия для определения того, как различные бизнес-объекты должны взаимодействовать друг с другом
для достижения необходимого результата. [7]

2.3.3 База данных

Система управления базами данных (СУБД) играет чрезвычайно важную роль в сети
разработка приложений в целом и как часть серверной части в частности. СУБД

предоставляет возможность сохранения, изменения и удаления данных. Это позволяет запрашивать
(извлекать) данные из фактической базы данных для дальнейшей обработки логикой приложения
слой. СУБД предоставляет способы определения того, как организованы данные, обеспечения различной безопасности
измеряет, поддерживает целостность данных, создает правила параллелизма и управляет ими, добавляет и отслеживает
пользователи и т.д. Существует два принципиально разных типа баз данных: реляционные и
не относящийся к отношениям.

Системы управления реляционными базами данных (СУБД) были абсолютно самыми
популярная база данных для использования более 20 лет [9]. Все базы данных отношений используют структурированные
Язык запросов (SQL) как язык для манипулирования данными и администрирования
база данных. Поэтому реляционные базы данных очень часто называют базами данных SQL. SQL
базы данных хранят данные в таблицах [9]. Внутри таблиц данные представлены в виде
строк и столбцов, где строки являются записями, а столбцы - атрибутами [9]. Эти данные являются
высокая структурированность и схема хранения должны быть очень строгими [9]. Разные таблицы поддерживают
определенный вид логических связей, называемых отношениями. Такого рода отношения создаются
на основе заданного взаимодействия между этими таблицами. Таким образом, именно здесь термин
Откуда взялась “реляционная база данных". Реляционные базы данных обычно используются в случае
где гибкость уступает строгой схеме и вертикальному подходу с более ре-
исходные коды для одного сервера в основном используются для достижения масштабирования [9]. Некоторые из самых популярных
СУБД - это: Oracle, MySQL, SQL Server, PostgreSQL и т.д.

Популярность нереляционных баз данных начала расти совсем недавно. По словам Ливитта Н .
(2010), одно из наиболее значительных повышений популярности нереляционных баз данных, произошедших
впервые в 2007 году, когда Amazon опубликовала работу, описывающую распределенный нереляционный
система баз данных под названием Dynamo, которая была создана для внутреннего использования [10]. Нереляционный
базы данных обычно не используют SQL в качестве языка для управления данными. Следовательно,
их очень часто называют базами данных NoSQL (не только SQL). Тип NoSQL
базы данных коренным образом отличаются от обычной системы управления реляционными базами данных
темы [9]. Сравнение табличного хранилища данных, используемого СУБД, базами данных NoSQL
хранит данные в различных форматах, таких как пары ключ / значение, графики, широкие столбцы,
Формат JSON или документа и т.д. Данные могут быть очень динамичными, а схемы хранения
являются полностью гибкими. В качестве преимущества такой гибкости каждая запись может иметь разные
свойства и новые свойства могут быть легко добавлены при необходимости без необходимости
изменить схему, используемую базой данных, поскольку схема диктуется не
база данных, но с помощью приложения [9]. Таким образом, базы данных NoSQL хороши в случаях

где данные являются сложными, вложенными или не очень структурированными, а масштабирование обычно об-
обеспечивается разделением данных на несколько серверов, что означает применение горизонтального подхода [9].

MongoDB, Apache Cassandra, Redis, Apache HBase и neo4j являются одними из наиболее

популярные базы данных NoSQL на данный момент.

3

Технологии

3.1

Node.js

Node.js является ли лицензированная MIT среда выполнения JavaScript с открытым исходным кодом, предназначенная для
создание огромного разнообразия масштабируемых серверных приложений. Он предназначен для асинхронного и
модель программирования, управляемая событиями, которая делает его эффективным и легким. [11] Node.js был
создан Райаном Далем в 2009 году.

3.1.1 Архитектура

Node.js построен поверх движка JavaScript версии 8. Тот самый, который используется в Google
Браузер Chrome. Движок V8 написан на C ++ и компилирует код JavaScript напрямую
к собственному машинному коду без необходимости интерпретировать код в режиме реального времени. Эта особенность
обеспечивает Node.js с возможностью очень быстрого выполнения кода. Но JavaScript версии 8
двигатель - это не единственный компонент, Node.js архитектура может быть продемонстрирована с помощью
рисунок ниже.

Рисунок 4. Node.js архитектура

Скопировано с Tsonev (2015) [12]

Как видно из рисунка 4, Node.js состоит из нескольких компонентов. А именно V8
движок, пул потоков libeio, который используется для выполнения операций асинхронного ввода-вывода (I/O)
ations и цикл событий libev в качестве основы, стандартная библиотека узла вверху и узел
привязки посередине [12]. Стандартная библиотека узла содержит основную функциональность и является
почти исключительно написан на JavaScript. Node.js привязки действуют как мост для соединения Ja-
Библиотеки vaScript и
операционная система
. Все компоненты, кроме верхней подставки JS-
библиотека ard написана на C / C ++, чтобы получить огромный прирост производительности.

3.1.2 Асинхронный и однопоточный

В традиционной модели сервера приложений параллелизм обеспечивается за счет использования блокирующего ввода-вывода
и несколько потоков - по одному потоку для каждого соединения. Каждый поток должен ожидать ввода-вывода
завершение перед обработкой следующего запроса. В Node.js , в свою очередь, имеет единственного исполнителя
ный поток, без какого-либо переключения контекста или ожидающего ввода-вывода [13]. Для любого запроса ввода-вывода про-
определены функции обработки, называемые обратными вызовами, которые впоследствии вызываются из
цикл событий, когда данные становятся доступными или происходит что-то еще важное.
модель цикла обработки событий и обработчика событий - это обычная вещь, вот как
скрипты, написанные на JavaScript, выполняются в браузере [13]. Ожидается, что про-
gram быстро вернет управление циклу событий, чтобы следующее задание в очереди могло быть

звонил. Чтобы проиллюстрировать это на примере, Райан Дал [13] спрашивает, что происходит при выполнении следующего кода:

```
результат = запрос (`выбрать * из T`);
// использовать результат
```

Листинг 1. Пример блокирующего кода

Скопировано из Даля (2010) [13]

В листинге 1 показан пример кода блокирующей модели программирования. И ответ на

Вопрос Райана Даля заключается в том, что в большинстве случаев программное обеспечение на этом

этапе приостановлено, что означает

, что оно просто вообще ничего не делает, в то время как уровень доступа к базе данных отправляет

запрос на
база данных, которая вычисляет результат и возвращает данные [13]. В зависимости от
сложность запроса, его выполнение может занять очень заметное время. Это плохо, будь-
причина в том, что пока поток простаивает, может прийти другой запрос, и если все потоки заняты,
запрос будет просто отклонен. В то же время переключение контекста не является бесплатным [13].
Чем больше потоков запущено, тем больше времени процессор тратит на сохранение и восстановление

их состояние [13]. Более того, стек выполнения каждого потока занимает место в
память [13]. И просто из-за асинхронного ввода-вывода, управляемого событиями, Node.js устраняет большинство
из этих накладных расходов, привнося лишь совсем немного своего.

Более того, реализовать параллелизм с использованием потоков может оказаться довольно сложной задачей. Причина-
причиной этой сложности является необходимость контролировать доступ к общим переменным и различным
стратегии предотвращения взаимоблокировок и конкуренции между потоками. Node.js , в свою очередь, использует
совершенно другой подход к достижению параллелизма. Обратные вызовы , которые запускаются из
цикл событий - это гораздо более простая модель параллелизма, как для понимания, так и для реализации.
размышление.

Чтобы проиллюстрировать необходимость асинхронного ввода-вывода, Райан Дал [13] напоминает об относительности доступа
раз к разным объектам. Доступ к объектам в памяти занимает порядка наносекунд
и это намного быстрее, чем доступ к объектам с жесткого диска или по сети
что, в свою очередь, исчисляется секундами и миллисекундами. [13] Время доступа к объектам
которые не хранятся в памяти, измеряется множеством тактовых циклов и могут быть
вечность, если клиент, не дожидаясь загрузки страницы в течение двух секунд, станет
надоело пялиться в окно браузера и перейду в другое место.

В Node.js упомянутый выше запрос должен быть написан следующим образом:

```
запрос (`выбрать * из T`, функция (результат) {
// использовать результат
}):
```

Листинг 2. Node.js пример неблокирующего кода

Скопировано из Даля (2010) [13]

Разница между примером блокирующего кода, продемонстрированным в листинге 1, и не-
блокировка, показанная в листинге 2, заключается в том, что в этом примере результат запроса не возвращается
как значение функции, но передается в качестве параметра функции обратного вызова, которая
будет вызван, когда будет доступен результат. Следовательно, происходит возврат к циклу событий
почти немедленно, и сервер может приступить к обслуживанию других запросов [13]. Один из

такие запросы будут ответом на запрос, отправленный в базу данных, а затем
будет вызвана функция обратного вызова. Такая модель немедленного возврата к циклу событий в-
увеличивает общее использование ресурсов сервера. И это здорово для владельца

сервер, но еще больше пользы получает пользователь, для которого содержимое страницы является
доступно намного быстрее.

3.1.3 Кластеризация

Как указывалось выше, единый Node.js сервер, прослушивающий определенный порт, работает только на
одном
нить. Следовательно, он не использует преимущества многоядерной системы, поскольку работает только
на
одноядерный. Следовательно, запуск одного Node.js процесс в многоядерной системе - это
пустая трата ресурсов. Использовать и извлекать выгоду из всех ресурсов многоядерной системы
Node.js имеет концепцию, называемую кластеризацией. [14]
Node.js кластеризация позволяет легко создавать отдельные процессы, которые все могут совместно
использовать
тот же порт сервера. Например, запуск Node.js HTTP-сервер на порту 80 в кластере
режим с 4 процессами означает, что на самом деле это 4 отдельных процесса, каждый из которых
прослушивает порт
80. И как Node.js имеет асинхронную архитектуру с мощным V8 под капотом
он обладает очень хорошей производительностью, которая может быть увеличена еще больше с помощью
кластеризация. С помощью всего лишь пары строк кода или просто изменив конфигурацию в
в случае использования другого программного обеспечения для управления процессами, такого как PM2,
производительность
может быть увеличен в несколько раз. Однако новая проблема может быть связана с общим повторным
источники, такие как база данных, которые могут стать узким местом в этой ситуации.
Node.js будет ли нагрузка балансировать запросы между процессами для повышения производительности
обычно в циклическом стиле. Кроме того, это позволяет добиться нулевого времени простоя в качестве
различных про-
процесс может завершиться сбоем, и запросы будут перенаправлены другому процессу, в то время как
этот процесс
выздоравливающий. [14] Кроме того, поддерживается нулевое время простоя во время обновлений
сервера.
Обновление сервера почти всегда требует перезапуска, который в случае только одного процесса
запуск вызовет некоторое время простоя, так как серверу требуется время для перезагрузки, но с
помощью
после кластеризации все процессы, запущенные в кластере, могут быть перезапущены один за другим
после
предыдущие уже начались.
Однако, при всех преимуществах и простоте Node.js кластеризация может быть сложной задачей-
иногда одалживаю. Потому что для успешного использования концепции сервер должен быть
государственным
меньше, другими словами, он не должен сохранять какое-либо состояние в памяти. Поскольку, это
должно быть
безразлично, к какому процессу в какое время поступает запрос. И создание сервера
отсутствие гражданства может быть проблемой. Однако, если это вызов, есть очень хороший способ

вокруг него. А именно, хранилище общей структуры данных в памяти, такое как Redis или Memory-
кэшированный. Таким образом, процессы могут использовать это общее хранилище для сохранения
некоторого состояния и по мере его
находится в памяти, это будет быстро.
Следовательно, Node.js кластеризация предоставляет такие мощные функции, как использование всех

ресурсы сервера, балансировка нагрузки и нулевое время простоя, но, конечно, с некоторыми трудностями-
такие проблемы, как отсутствие состояния или сохранение общего состояния.

3.1.4 Производительность

Как указано выше, Node.js асинхронная и однопоточная модель делает это очень быстро.
И для подтверждения этого утверждения сравнение производительности и оценка Node.js ,
Веб-технологиями PHP и Python руководили Кай Лей, Чжи Тан и Инин Ма ин
2014 [15]. Исследование проводилось с использованием сценарных тестов, имитирующих реалистичные поведение пользователя и контрольные тесты. Модуль бенчмаркового тестирования состоял из трех основных
части: обслуживание веб-страницы “Hello World”, выполнение операции выбора из базы данных и вычислять
определение значений чисел Фибоначчи. [15]

Результаты первого теста обслуживания веб-страницы “Hello World” можно увидеть на следующей странице:
мычащая фигура:

Рисунок 5. Среднее количество запросов в секунду для теста “Hello World”

15 (45)

Скопировано с Кай Лэя, Чжи Тана и Инин Ма (2014) [15]

Результаты, показанные на рисунке 5, основаны на двух параметрах: количестве запросов на секунда по вертикальной оси и количество одновременных подключений пользователей по горизонтальной
ось. Чем выше цифры, тем лучше. Следовательно, основываясь на рисунке 5, можно четко
видел это Node.js демонстрирует лучшие показатели среди всех трех конкурентов.

Когда число одновременно работающих пользователей довольно невелико - от 10 до 100, Node.js и PHP показывают довольно близкие результаты с частотой около 3100 запросов в секунду (req / s) и 2600 повторений в секунду
соответственно. Однако, когда число пользователей приближается к 200, а затем и к 1000,
Производительность PHP резко падает примерно на 200-600 запросов в секунду по сравнению с Node.js который по-прежнему удерживает лидерство с показателем более 2500 рекв /с. Python, в свою очередь, выполняет
стабильно на протяжении всего теста с частотой около 300-500 req/c.

Во втором контрольном тестировании сравнивается производительность выбранных веб-технологий в зависимости от скорости операции выбора базы данных. Результаты показаны на рисунке ниже:

Рисунок 6. Среднее количество запросов в секунду для теста операции выбора базы данных

Скопировано с Кай Лэя, Чжи Тана и Инин Ма (2014) [15]

Результаты тестирования, которые можно увидеть на рисунке 6, основаны на том же параметре-
ters как результаты
теста “Hello World”, показанного на рисунке 5, а именно количества запросов

16 (45)

в секунду и количество одновременных подключений пользователей. Как показано на рисунке 6, Node.js имеет наилучшую производительность около 2600-3200 рекв/с, когда количество кон-
текущие подключения пользователей составляют менее 500 и около 1800 рекв/с при 1000 подключениях пользователей
связи. PHP сохраняет задержку в 40-60% при примерно 1400-1700 запросах в секунду до 200 одновременных
пользователи. Но разрыв между PHP и Node.js становится почти в два раза больше с
500 и 1000 пользователей. Python поддерживает стабильную производительность около 200 запросов в секунду во всем
испытание.

Последний тест измеряет производительность Node.js , PHP и Python-Веб с использованием

Числа Фибоначчи. Тест основан на скорости выполнения запросов на вычисление
обозначение значений последовательностей из 10, 20 и 30 чисел Фибоначчи. Результаты
эталонный показатель можно найти в следующей таблице:
Таблица 1. Результаты для вычисления 10/20/30 чисел Фибоначчи
Скопировано с Кай Лэя, Чжи Тана и Инин Ма (2014) [15]
Результаты, представленные в таблице 1, сравниваются с использованием количества запросов на
секунда и время на каждый запрос. Согласно таблице 1, лучшее время среди выбранных
технологии веб-разработки имеют Node.js . PHP с частотой около 2000 запросов в секунду и 0,401мс
время на каждый запрос показывает почти одинаковую производительность при вычислении, когда
Число Фибоначчи 10 как Node.js со скоростью почти 2500 запросов в секунду и временем 0,401 мс
соответственно
тивно. Python, в свою очередь, отстает почти втрое. Однако, когда расчеты будут-
более сложным является разрыв в производительности между PHP и Node.js резко увеличивает
калли и PHP начинают работать почти на том же уровне, что и Python, обрабатывая только с
от 2 до 3 запросов в секунду с числом Фибоначчи 30. Node.js , в то же время, демонстрирует способность
к
обрабатывает почти 60 запросов в секунду.
Основываясь на результатах, проведенных в ходе исследования, можно ясно видеть, что Node.js
выполняет
лучше и может обрабатывать гораздо больше запросов в определенные моменты, чем PHP и Python-
паутина [15]. Тесты доказывают , что Node.js является идеальным выбором для интенсивного ввода-
вывода с высокой согласованностью
удобные и масштабируемые веб-приложения с их легкостью и эффективностью.
17 (45)
3.1.5 Диспетчер пакетов узла (NPM)
NPM - самая популярная система управления пакетами JavaScript и распространения,
который
является Node.js менеджер пакетов по умолчанию и крупнейший в мире реестр программного
обеспечения [16].
Концептуально он похож на такие инструменты, как apt-get в Debian, rpm или yum в Redhat и
Fedora, MacPorts на Mac OS X,
CPAN - это Perl, а PEAR - PHP. Его задача состоит в том, чтобы обеспечить
публикация и распространение Node.js модули через Интернет с помощью простого com-
интерфейс командной строки. NPM позволяет быстро находить пакеты для решения конкретной задачи,
загружайте и устанавливайте их, а также управляйте уже установленными модулями и необходимыми де-
ожидания.
Все модули NPM хранятся в “node_modules” [17]. Требуемый
зависимости для проекта вместе с номерами версий каждого из них перечислены в
файл “package.json”. Находятся как папка “node_modules”, так и файл “package.json”
в корневом каталоге проекта.
С помощью файла “package.json” в этом нет необходимости
для загрузки

на сервер все модули в папке “node_modules”, требуется

загрузить только файл “package.json”, а затем запустить “npm install” с помощью командной строки,

чтобы
установите все необходимые модули. Следовательно, это делает Node.js проекты довольно легкие
а также легко делиться. [17]
3.2
Экспресс
Express - это минималистичный и гибкий веб-фреймворк для Node.js приложения, которые про-
предоставляет обширный набор функций для мобильных и веб-приложений. Это про-

содержит множество служб HTTP-модулей и обработчиков промежуточного программного обеспечения, что делает создание
создание надежного и безотказного API гораздо более простой и быстрый способ, чем использование основного узла
модули. Более того, все методы и функции, предоставляемые экспресс-модулем, делают
не скрывать ни одного из родных Node.js функции. Таким образом, можно работать с
то же самое Node.js Основные объекты HTTP при использовании Express. [18] Экспресс состоит из трех
основные компоненты: маршрутизатор, маршрутизация и промежуточное программное обеспечение.
Маршрутизатор является основным экземпляром экспресс-модуля, который используется для указания маршрутов для
HTTP-запросы, настройка промежуточного программного обеспечения, настройка механизма шаблонов и т.д. [18]. Это может быть кре-
выполняется следующим образом:


```
var express = требовать('экспресс');
var app = экспресс();
```

Листинг 3. Экземпляр экспресс-маршрутизатора
Скопировано из Express [18]

Как видно из листинга 3, объект Express router может быть создан путем запроса
экспресс-модуль и вызов основной функции “express()”. Экземпляр маршрутизатора обычно вызванный
“приложение” и содержит все функциональные возможности, предоставляемые Express.
Маршрутизация определяет, как приложение должно реагировать на запрос клиента, который был
отправлено по определенному адресу, названному конечной точкой или маршрутом [18]. Определение маршрута имеет
следующая структура:

приложение.МЕТОД(ПУТЬ, ОБРАБОТЧИК)

- приложение является экземпляром express.
- МЕТОД - это метод HTTP-запроса в нижнем регистре.
- ПУТЬ - это путь на сервере.
- ОБРАБОТЧИК - это функция, выполняемая при выполнении маршрута соответствует.

Листинг 4. Структура определения маршрута
Адаптировано из Express [18]

В листинге 4 показано, что каждый маршрут состоит из URI или пути и конкретного HTTP-запроса
метод, например, GET, POST и т.д. Каждый маршрут может иметь минимум один хан-
dler, который выполняется при достижении конкретной конечной точки.
Экспресс
обработчики промежуточного программного обеспечения - это функции, которые имеют доступ к объекту ответа
(res), объект запроса (req) и к следующей функции промежуточного программного обеспечения в request-re-
спонтанный цикл подачи заявки [18]. Следующая функция промежуточного программного обеспечения обычно обозначается
переменная, которая называется “next”. Функции промежуточного программного обеспечения могут выполнять любой код, изменять
объекты запроса и ответа, завершают цикл запрос-ответ отправкой ответа

и функции промежуточного программного обеспечения могут вызывать следующую функцию промежуточного программного обеспечения [18]. Если запрос-повторный
спонтанный цикл не завершается текущей функцией промежуточного программного обеспечения, она всегда должна передавать управление
к следующей функции промежуточного программного обеспечения
вызвав “next()”. В противном случае запрос зависнет.
Все три основных элемента Express можно увидеть на следующем рисунке:
Рисунок 7. Экспресс-основные компоненты с описанием промежуточного программного обеспечения
Скопировано из Express [18]
Как показано на рисунке 7, для первой строки требуется модуль Express, для второй строки
создает основной экземпляр “приложения” Express Framework. Следующая строка - это пример маршрут к
root “/” местоположение с промежуточной функцией, которая ничего не делает и просто вызывает следующая функция промежуточного программного обеспечения. Код, написанный в последней строке, запускает сервер для прослушивания
порт 3000 для любых входящих запросов.
В то же время Express имеет интеграцию с шаблонизатором, которая в значительной степени помогает сделать
приложение стало более гибким и динамичным. Использование шаблонизатора делает это намного проще разделить логику приложения и представления. С помощью движка шаблонов можно используйте статические файлы шаблонов с переменными. Эти переменные заменяются фактическими значениями в
время выполнения шаблонизатором, а затем файл шаблона преобразуется в HTML
файл, который передается клиенту [18]. Также популярны EJS, Усы, Мопс и Нефрит
шаблонизаторы, интегрированные для работы с Express [18]. Такого рода подход
с рендерингом HTML-файлов во время выполнения из статических шаблонов с поддержкой переменных делает дизайн веб-страниц намного более простым и удобным процессом, чем, например,
создание HTML-файлов и их содержимого путем конкатенации строк.

3.3 MongoDB
MongoDB - это кроссплатформенная база данных NoSQL, ориентированная на документы с открытым исходным кодом, которая
фокусируется на гибкости и масштабируемости при одновременном обеспечении высокой доступности и производительность. [19]
3.3.1 Архитектура и концепции
MongoDB использует концепцию пар ключ-значение и формат, в котором хранятся данные
является BSON (двоичный JSON) [20]. Сравнивая терминологию и концепции MongoDB и
в любой базе данных SQL можно выделить несколько основных отличий. Прежде всего, Пн-goDB концептуально ничем не отличается от обычной и всем знакомой базы данных SQL.
Внутри MongoDB может быть ноль или более баз данных, каждая из которых является контейнером для другие организации [20]. Во-вторых, база данных может содержать ноль или более "коллекций". Кол-лекция настолько похожа на традиционную "таблицу", что их можно смело считать одинаковыми.
В-третьих, коллекции состоят из нуля или более "документов". И в этом случае документ
может рассматриваться как "строка". Документ представляет собой объект JSON с обязательным пред-значение идентификатора объекта, которое MongoDB автоматически устанавливает и контролирует его уникальность, вплоть до
уникальность во всем мире [20]. В-четвертых, документ состоит из одного или нескольких
"поля", которые очень похожи на "столбцы". В-пятых, "индексы" в MongoDB почти
идентичен таковым в реляционных базах данных.
Наконец, в MongoDB есть понятие “курсоры”
которые отличаются от предыдущих пяти пунктов [20]. Важно понимать , что

когда данные запрашиваются из MongoDB, он возвращает курсор, с помощью которого про-
grammer может делать что угодно, например, подсчитывать или пропускать определенное количество предыдущих записей без
загружаем данные сами [20]. Суммируя указанные выше моменты, MongoDB подтверждает
системы "баз данных", которые состоят из "коллекций". "Коллекции" состоят из "документов".
Каждый "документ" состоит из "полей". "Коллекции" могут быть проиндексированы, что улучшает самоконтроль
производительность сборки и сортировки. Наконец, получение данных из MongoDB сводится к получению-
установите "курсор", который выдает данные по мере необходимости.
Базовый пример документа в MongoDB можно увидеть в следующем списке:


```
{
_id: идентификатор объекта("5099803df3f4948bd2f98391"),
имя: {имя: "Алан", фамилия: "Тьюринг" },
рождение: новая дата ("23 июня 1912"),
смерть: новая дата("07 июня 1954"),
вклады: [ "Машина Тьюринга", "тест Тьюринга", "Туринджеры" ],
просмотров: NumberLong(1250000)
}
```

Листинг 5. Пример документа MongoDB
Скопировано из MongoDB [20]
Как показано в листинге 5, документы в MongoDB состоят из ключа или поля
– пары значений.
тип значения варьируется и может быть любого типа из типов данных BSON, а также других
документы, включая массивы документов. Например, в листинге 5
поле “имя” имеет
введите документ и

содержит другие поля, в частности, “первое” и “последнее”. Поля “рождение”

и “смерть” имеют тип Дата. Массив строк хранится в поле, которое называется “contributs”. И

поле “просмотры” содержит данные типа NumberLong. При этом присутствует обязательный уникальный

идентификатор “_id” типа ObjectId.

3.3.2 Динамическая схема

Одной из ключевых особенностей баз данных NoSQL и MongoDB, в частности, является динамичность
схемы. Документы внутри одной коллекции могут иметь совершенно разную структуру с
поля различных типов. Схема задается не на уровне базы данных, а на
уровень приложения, означающий, что новые поля или документы с другой структурой могут быть
добавляется всякий раз, когда это необходимо, даже во время выполнения. Разработчикам нет необходимости
определите схему заранее. Эта функция особенно полезна для улучшения или расширения
в приложении, поскольку новые функции могут быть легко добавлены без каких-либо простоев и с
нет времени, потраченного на переопределение структуры.

3.3.3 Репликация

Репликация - это способ обеспечить избыточность и получить высокий уровень доступности данных.
Ответ-
катион - это все о наличии нескольких копий данных, вероятно, на разных машинах, которые

защищает данные от сбоев и потери данных на одном компьютере. Это также очень полезно для
резервное копирование и восстановление в аварийных ситуациях. [20]
MongoDB предлагает репликацию с помощью наборов реплик. Набор реплик может иметь несколько
узлы, которые содержат данные, и узел-арбитр в качестве необязательного. Арбитр поддерживает
кворум среди узлов в наборе реплик установлен и не хранит никаких наборов данных. Один элемент

данных
несущие узлы выбираются в качестве основного узла, в то время как все остальные узлы являются вторичными узлами.
[20] Все операции записи принимаются основным узлом, а затем передаются на
вторичные, так что данные отражают данные, хранящиеся в первичном узле. В
в случае сбоя основного узла или если он становится недоступным, будет избран новый основной среди вторичных узлов. Концепция репликации MongoDB может быть визуализирована с помощью следующий рисунок:
Рисунок 8. Схема репликации MongoDB
Скопировано из MongoDB [20]
Исходя из рисунка 8, видно, что сервер с логикой приложения отправляет запись
и запросы на чтение к первичному узлу, одновременно распространяя записи на вторичный
узлы, что приводит к синхронизации данных между всеми узлами.

Репликация повышает отказоустойчивость и в некоторых случаях увеличивает пропускную способность для чтения
система, которая весьма важна в производственной системе. Система увеличивает доступность данных-возможность и создает параметры восстановления, используя такую функцию MongoDB, как репликация.
3.3.4 Сегментирование
Когда размер данных становится очень большим, MongoDB помогает решить эту проблему с помощью помощь в шардинге. Масштабирование системы по горизонтали путем распределения наборов данных по несколько серверов называется сегментированием. Таким образом, каждая машина выполняет только часть работы-
загрузка и подмножество данных, что приводит к повышению производительности по сравнению с одиночными
сервер, который обрабатывает все данные и рабочую нагрузку [20]. Пример общей коллекции может видно на рисунке ниже:
Рисунок 9. Пример сегментирования MongoDB
Скопировано с [https://docs.mongodb.com/manual]
Рисунок 9 демонстрирует концепцию сегментирования. Существует две коллекции, одна из которых хранится полностью в одном сегменте, но другой разделен на два сегмента.
Таким образом, сегментирование обеспечивает распределенное по разным машинам чтение и запись рабочая нагрузка. Емкость хранилища увеличивается по мере того, как каждый дополнительный фрагмент увеличивает емкость

вся система. Доступность данных становится выше по мере того, как сегментированный кластер способен выполнять пар-
tial читает и записывает, в то время как некоторые другие осколки могут быть отключены. [20] Цена, потенциально,
также может быть ниже, поскольку обычно используется большее количество машин средней производительности и мощности.
союзник дешевле, чем обновлять уже мощный сервер.
3.3.5 Мангуст
Проверка и приведение данных могут быть довольно сложными с MongoDB, однако есть Mon-
гусь, который был создан для того, чтобы сделать это очень простым. Мангуст обеспечивает очень легкий
и готовый способ моделирования данных приложения с использованием решений на основе схемы.
[21] Используя схемы, можно определить на требуемом уровне, какого рода данные должны находиться в конкретном документе MongoDB. Таким образом, становится намного проще выполнять кастинг, валидацию
и создайте шаблонную часть бизнес-логики. Более того, Mongoose предоставляет методы для создавайте подключения к MongoDB и управляйте ими.

3.3.6 Плюсы и минусы

У MongoDB есть много положительных моментов, таких как простые и мощные JSON-подобные данные схема, довольно гибкий язык запросов, динамические запросы, полная поддержка индексов, очень быстрый
обновления, эффективное хранение больших двоичных данных, ведение журнала операций, которые изменяют данные,
поддержка отработки отказа и масштабируемости, а также MongoDB могут работать в соответствии с Парадигма MapReduce. Все это делает MongoDB очень хорошим выбором для таких вещей
как потоки данных большого объема, таргетинг рекламы, мониторинг социальных сетей, большое количество
метаданные, новости и различные приложения для управления контентом и т.д. Тем не менее, это также имеет свои недостатки, такие как отсутствие оператора ‘join’, который может затруднить организацию передача данных иногда довольно сложный процесс, отсутствие такого понятия, как транзакция, поэтому это
невозможно объединить несколько операций в одном атомарном действии. Гарантируется только атомарность
на уровне всего документа, т.е. частичное обновление документа не может произойти.

Таким образом, MongoDB может быть плохим выбором для использования, например, в банковской системе
поскольку существует множество объединенных операций, которые должны быть атомарными.

4

Реализация

4.1

Описание проекта

Проект представляет собой веб-приложение, которое было разработано для облегчения процесса создания,
управление, регистрация и оплата танцевальных турниров. Разработанная платформа
предоставляет часть всех необходимых функций. Тем не менее, можно легко расширить функциональность
добавлено в приложение. На момент написания этой статьи следующие интерфейсы
были реализованы:

- Интерфейс администратора для создания учетных записей и управления ими для различных организаций-
организации, которые организуют турниры.
- Интерфейс администратора для чата поддержки.
- Интерфейс организации для просмотра и обновления контактной и бизнес-информации размышление.
- Интерфейс организации для просмотра и обновления платежных учетных данных.
- Организационный интерфейс для создания созданных турниров и управления ими с помощью очень индивидуальные настройки.
- Организационный интерфейс для создания полей формы и групп форм для участников брюки для заполнения в рамках процесса регистрации.
- Интерфейс организации для создания и просмотра созданных шаблонов электронной почты с помощью возможность вставлять подстановочные знаки, которые впоследствии могут быть автоматически

заменены на
данные соответствующего участника турнира.
•
Организационный интерфейс для чата поддержки.
•
Интерфейс участника для просмотра всех доступных турниров и с возможностью
подписаться на избранного.
Как видно из приведенного выше списка, приложение состоит из трех основных частей:
интерфейсы администратора, организации и участника.
4.2
Настройка среды
Приложение было разработано на настольном компьютере под управлением операционной системы
Windows 10.
Помимо ПК и ОС для реализации требовалась среда разработки, чтобы

быть настроенным. Среда разработки состоит из различных программных средств и
инструменты для работы с ними. Таким образом, было установлено следующее программное
обеспечение:
Node.js
Node.js является неотъемлемой частью процесса разработки этого проекта, поскольку он обеспечивает
среда выполнения для выполнения кода JavaScript. Его можно загрузить с
официальный сайт (https://nodejs.org/en/download /). После установки самый простой способ
чтобы проверить, работает ли он правильно, нужно запустить команду node -v в терминале. Ком-
mand печатает установленную версию Node.js . Версия 6.3.1 использовалась во время де-
развитие.
MongoDB
MongoDB является второй по важности частью среды разработки, потому что
приложению требуется база данных для хранения различных данных и запросов к ним. Он может быть
установлен
с официального сайта (https://www.mongodb.com/download-center). После установки-
тион, сервер MongoDB можно запустить с помощью команды mongod из терминала.
При разработке использовалась версия 3.2.3.
Веб - шторм
WebStorm - это очень мощная интегрированная среда разработки (IDE) для JavaScript
развитие. Он имеет встроенную поддержку для Node.js проекты. А также завершение кода,
мощные функции навигации, обнаружение ошибок "на лету" и функции рефакторинга, он имеет
удобный интерфейс для отладки Node.js приложения.
Робомонго
Robomongo - это инструмент управления MongoDB с очень удобным пользовательским интерфейсом и
на-
полная поддержка оболочки MongoDB.
Исходное дерево
SourceTree - это очень мощный графический интерфейс Git с отличным и простым интерфейсом.

4.3
Структура и реализация проекта
Структура проекта четко определена, и компоненты разделены соответственно
их роли. Обзор структуры можно увидеть на следующем рисунке:
Рисунок 10. Обзор структуры проекта в интерфейсе администратора
На рисунке 10 показана структура приложения. Точка входа для приложения-
тион - это “app.js ” файл. Он содержит всю инициализацию, конфигурацию, маршрутизацию и сервер

начало
логика. Бизнес-логика находится в каталоге “контроллер”. Все модели
соответствующий
коллекции баз данных расположены внутри каталога “модель”.

Каталог “просмотр” содержит компоненты представления, которые показываются пользователю приложение. Все изображения, CSS, шрифты, картинки и JS-код, используемые во внешнем интерфейсе, лежат

в каталоге “общедоступный”. Различные служебные функции находятся в каталоге “system”.

А файл “package.json” содержит список всех зависимостей, которые используются в проджект.
Как было указано выше, “app.js ” является основным и вводным файлом в приложении. Прежде всего, это считывает и инициализирует все модели в каталоге моделей. Затем он требует и инициализирует экземпляры всех контроллеров, расположенных в контроллере
папка. Затем создается экземпляр “ex-

press” и выполняется подключение к базе данных с использованием модуля “mongoose”
установлено. После этого все необходимые конфигурации
сделаны для экземпляра “экспресс”.

Затем с помощью модуля под названием “паспорт” создается простая система аутентификации сеанса создается. Далее, обработчики для запросов GET и POST и отдельно для
маршрут “/login” создается с использованием экземпляра “express”. И, наконец, сервер “экспресс” запускается на порту, который указан в конфигурации.
Следующая наиболее важная часть приложения - это бизнес-логика. Контроллеры , которые являются определенные в каталоге “controller”, отвечают за всю бизнес-логику, которая происходит в применение. Каждый контроллер имеет одинаковую структуру и должен расширять основной контроллер это называется “AppController”. Он содержит логику для разбора и подготовки запроса
информация, которая должна быть легко доступна каждому из дочерних компонентов. А также наличие логика инициализации и подготовки, “AppController” имеет общие методы, которые часто
используется производными классами.
Как было рассмотрено выше, каталог моделей содержит модели, которые являются абстракцией поверх соответствующих коллекций MongoDB. Каждая модель должна расширять класс, который является
называется “AppModel”. Родительский класс содержит логику для создания экземпляра Мангуста модель с соответствующей схемой. Более того, общие функции создания, чтения, обновления, деметоды lete (CRUD) для работы с коллекцией также определены в “AppModel”.
Просмотры - это последняя часть приложения. Каждое представление представляет собой шаблон EJS с определенными
HTML-разметка и JavaScript-код для вставки данных в шаблон. Некоторые файлы EJS
содержат небольшой компонент, созданный для определенной страницы, в то время как часть представлений определяет

общий макет, который используется во всем приложении. Компоновка, корпус и все необходимые Шаблоны EJS объединяются в окончательный вид перед отправкой клиенту. Таким образом, пользователь приложения может видеть полную HTML-страницу с необходимыми компонентами.

4.3.1 Поток запросов-ответов

Поток запросов-ответов в приложении, описанный в этой статье, основан на cus-
том фреймворк. Было решено создать пользовательский фреймворк
поверх стандартного

маршрутизатора “ex-

press” для лучшего контроля и гибкости. Таким образом, каждый запрос GET и POST перехватывается соответствующим обработчиком, а затем URL анализируется для определения исправьте контроллер и его метод обработки запроса. URL-адрес должен быть в форме: /{имя контроллера}/{метод}

Листинг 6. Шаблон URL

Как указано в листинге 6, каждый URL-адрес запроса должен содержать имя контроллера и правильный метод. Следовательно, после синтаксического анализа URL-адреса проверяется список всех контроллеров, если он имеет запрошенный контроллер. Если контроллер отсутствует в списке, статус 404 будет отправить клиенту. С другой стороны, если контроллер присутствует, запрос и повторное объекты sponse подготавливаются и добавляются в экземпляр контроллера, а затем запрошенный метод вызывается либо с параметрами GET, либо POST. Следовательно, внутри вызываемый метод обеспечивает доступ к объектам запроса, ответа и запрашиваемому параметры. А также объекты запроса и ответа, модель по умолчанию с тем же имя в качестве контроллера доступно в области действия метода. Во время выполнения метода-cution вся необходимая бизнес-логика, такая как чтение или запись в базу данных и обработка данных выполняется обработка. В конце выполнения можно отправить ответ только с данными или визуализировать необходимую страницу или определенный элемент с данными, используя адаптированная система просмотра и отправки обратно HTML-разметки.

Таким образом, созданная платформа обеспечивает очень легкую, гибкую и прозрачную разработку процесс разработки. Очень удобно добавлять дополнительные контроллеры, модели и представления. Использование этот фреймворк, приложение может быть легко расширено дополнительными страницами, информацией и функциональность.

4.3.2 Контроллеры

В этом разделе более подробно рассматривается и описывается роль каждого представленного контроллера в самой большой части проекта, которая является организационным интерфейсом.

Контроллер шкафа

Контроллер кабинета отвечает за сохранение и извлечение организационной информации, такой в качестве платежных данных, контактной и деловой информации. Есть несколько POST и GET методы, представленные в этом контроллере. Прежде всего, база метод “индексации”:

```
индекс() {
        this.redirect("/кабинет/панель управления");
}
```

Листинг 7.

“Индексный” метод управления шкафом

Как видно из листинга 7, метод “index” перенаправляет на относительный URI “cabi-сеть/панель управления”. Метод перенаправления объявлен внутри “AppController” и доступен поскольку каждый контроллер ext завершает “AppController”. Метод вызывает функцию перенаправления повторного ответный объект. Перенаправленный URI вызывает вызывает тот же метод “приборной панели” контроллер, который показан ниже:

```
панель управления(){
        this.render({ __заголовок: "Панель мониторинга"});
}
```

Листинг 8.
Метод “приборной панели” контроллера шкафа

В листинге 8 показана реализация метода “приборной панели” контроллера шкафа.

Метод вызывает функцию “render” для экземпляра класса. Метод “Render”, который отменен

штрафом в “AppController”, подготавливает необходимые параметры и отображает представление,

соответствующее

методу “dashboard”. Представление содержит интерфейс панели мониторинга со статистическими

данными
обзор тиков. Во-вторых, существует три способа сохранения информации об организации:
“saveContactInfo”, “saveBusinessInfo” и “savePaymentCredentials”. У всех них есть
та же структура:
31 (45)

```
Сохранить контактинфо(){
this.loadModel("Пользователь");
this.User.обновить(
{_id : этот.user._id},
{"$set" : {"contact_info" : this.request_data}},
(ошибка) => {
this.sendStatus(ошибка? 200 : 500);
                        });
}
```

Листинг 9.
Метод “saveContactInfo” контроллера шкафа
В листинге 9 показан фрагмент кода метода “saveContactInfo”. Первая строка в
метод вызывает
Метод “loadModel” “AppController” для подготовки запрошенного
смоделируйте и добавьте его в экземпляр класса.

Вторая строка вызывает метод “update”

загруженной модели “User”. Первый аргумент - это запрос, означающий, какие документы в
коллекция должна быть обновлена. Второй аргумент содержит полезную нагрузку обновления.
И третий и последний аргумент - это функция обратного вызова, которая будет вызвана после обновления либо завершается успешно, либо возвращает ошибку. Обратный вызов проверяет, возвращена ли ошибка, и отправляет

соответствующий статус клиенту. Метод “SendStatus” является абстракцией над

методом объекта ответа “sendStatus”, созданным “express”. Наконец, для отображения
сохраненные данные, два похожих

используются методы: метод “info” для получения и отображения контактной

и бизнес-информации и метод “paymentCredentials” для отображения представления с помощью рау-
измените учетные данные. Оба метода содержат только одну строку:

```
Платежные кредиты(){
this.найдИтеAndRender(
{_id : этот.user._id},
"Пользователь",
{ __заголовок : "Платежные данные"},
"пользователь"
        );
}
```

Листинг 10.
Метод “paymentCredentials” контроллера кабинета

Пример кода, продемонстрированный в листинге 10, показывает реализацию метода “платеж-

учетные данные”. Как можно видеть, метод вызывает только метод “findOneAndRender”

, который определен в “AppController”. Метод и абстракция выше запроса
данные из базы данных и визуализация конкретного представления с этими данными. Первый аргумент-
ment - это запрос к базе данных. Второй - это название модели, которое следует запросить.
Следующий аргумент содержит дополнительные данные, которые будут переданы для просмотра. И название
представления передается в качестве последнего аргумента методу “findOneAndRender”.
32 (45)
Контроллер полей формы
Контроллер полей формы имеет логику для отображения и сохранения полей формы, созданных
пользователь, а также формирование и отображение групп созданных полей формы. Созданный
поля формы и группы полей формы используются при создании турниров и шаблонов электронной
почты.
Следующие методы создают логику контроллера. Метод “Index” перенаправляет на
мето

d называется “показать”, который отображает интерфейс для создания и отображения полей

формы.

Создание полей формы поддерживается методом “сохранить”, который подготавливает и

сохраняет нужное поле формы. И, конечно же, существует метод “удалить” для удаления cre-
измененное поле формы. Большая часть логики в контроллере занята обработкой формы
полевые группы. Прежде всего, метод “группы” находит все ранее созданные группы форм
и поля формы и отображает представление для создания групп из созданных полей формы и
отображение обзора найденных. С помощью метода “getGroup” можно поз-
возможность получить представление с подробной информацией о запрашиваемой группе форм.
Создавать и
для удаления групп используются следующие методы
необходимо: “Сохранить группу” и “Удалить группу”
соответственно.
Контроллер электронной почты
Контроллер электронной почты отвечает за сохранение и отображение шаблонов электронной почты,
которые могут быть созданы
использование модульного редактора. Также можно вставить подстановочные знаки в шаблон, которые
были
образовано из th

e созданные поля формы. Первый и основной метод - это “индекс”. Он перенаправляет

на

метод “myTemplates” для отображения представления, содержащего список всех шаблонов электронной

почты, которые
были созданы пользователем. С этой точки зрения можно либо удалить шаблон с помощью

с помощью метода “removeTemplate” или отредактируйте шаблон. Если выбран параметр редактировать,

страница будет перенаправлена на метод “templateEditor”, который отображает представление с помощью

редактора, и с помощью метода “getTemplate” данные выбранного шаблона загружаются в
клиент. Представление редактора также содержит список подстановочных знаков, которые можно
использовать при создании
различные шаблоны. Чтобы получить список подстановочных знаков, используется метод “getWildcards”.
И, наконец,
новый шаблон или отредактированный существующий шаблон можно сохранить в базе данных с

помощью
Метод “saveTemplate”.
Контроллер входа
33 (45)
Контроллер входа
имеет только два метода. “Индекс”, который отображает страницу входа и “выход из системы”
это выводит пользователя из системы и перенаправляет на страницу входа в систему. Функция входа в систему, сеанса и выхода из системы-
особенности
предоставляются через пакеты NPM: “паспорт”, “экспресс-сессия” и “паспорт-lo-
кэл
”. На странице входа в систему после ввода имени пользователя и пароля появляется сообщение с призывом “войти”
конечная точка создана. Конечная точка создается с помощью
“экспресс” и “паспорт”, без контролера
есть таки
участие в авторизации. Обработчик, зарегистрированный в конечной точке “login”, проверяет
имя пользователя и пароль, и в случае правильных учетных данных создается сеанс входа в систему.
Выход из системы производится с использованием “паспортного” метода “logout”, который прикреплен к объекту запроса
по последующим запросам.
Контроллер поддержки
Контроллер поддержки отвечает за рендеринг страницы поддержки, сохранение и получение сообщений

мудрецы. Основной метод “индексирования” находит все разговоры в базе данных и отображает

страницу со списком разговоров. Отображаемая страница вызывает метод “сообщений”
с идентификатором первого разговора. Метод запрашивает базу данных для всех сообщений
в конкретном разговоре и возвращает отрисованный список, который затем отображается в
браузер.
Последний метод, представленный в контроллере поддержки, - “saveNewMessage”. Как
как можно догадаться по названию, он отвечает за сохранение сообщений в базу данных.
Турнирный Контролер
Турнирный контроллер содержит логику для поиска, редактирования, создания и удаления тура-
указания. Базовый метод “index” перенаправляет на метод “manage”, который отображает представление для списка турниров с полями поиска
. Когда страница открылась на клиенте, метод “get”
вызывается. По умолчанию метод находит все турниры, созданные пользователем, и возвращает
предоставленный список клиенту. Один и тот же метод используется с разными параметрами, когда
ищите конкретные турниры, используя поля поиска. Пользователь может нажать на любую турну-
щелкните в списке, чтобы получить подробную информацию. Информация запрашивается с помощью
метода “детализации”. Он получает данные, а затем отправляет клиенту визуализированное
представление подробных
информация о запрашиваемом турнире. Вид с подробностями имеет четыре кнопки для управления
возраст выбранного турнира. Переключить активный статус турнира можно с помощью

Метод “toggleActive” для переключения видимости турнира с помощью метода “toggleVisible”

. Турнир может быть удален с помощью метода “удалить”. Последняя кнопка повторно
отвечает за перенаправление на страницу редактора для редактирования конкретного турнира. Редактор
34 (45)
страница используется как для создания, так и для редактирования
турниры. Отвечает метод “Создать”
брат для рендеринга редактора турниров. Если идентификатор турнира будет передан в

метод, представление отображается с информацией о выбранном турнире. Поперек,
представление отображается с пустыми полями. Т
последним, но не менее важным, является метод “сохранить”, который
отвечает либо за сохранение созданного турнира, либо за отредактированный.

4.3.3 Модели

Каждая модель расширяет основной класс “AppModel”. Родительский класс имеет абстрактный метод функции для каждой операции с базой данных, такие как find, findOne, distinct, create, insert, update и т.д. Следовательно, контроллеры выполняют запросы к базе данных только с использованием методов определено в “AppModel”. Например, метод, отвечающий за вставку документов
доступ к базе данных можно увидеть ниже:

```
вставка(документы, обратный вызов){
пусть данные = документы;
если(!Массив.isArray(данные)){
данные = [данные];
}
        this._model.collection.insert(данные, обратный вызов);
}
```

Листинг 11.
метод “insert” основного класса модели

Листинг 11 демонстрирует одну из абстракций
представлен в классе “AppModel”. Как могу
видно, что метод принимает два параметра: документ или массив документов, которые необходимо вставить в базу данных и выполнить обратный вызов, который будет вызван с результатом данные после завершения операции. Первые пару строк в методе проверяют, не предоставлено ли переменная documents - это отдельный документ или массив, и если переменная не является массивом преобразует его в массив с одним элементом. Последняя строка
вызывает метод “insert”
с предоставленными данными и обратным вызовом для соответствующей коллекции с использованием мангуста
модель. Модель Мангуста создается с использованием метода “_initiate” в “AppModel”. Тело
часть метода инициализации заключается в следующем:

```
_initiate(схема, имя_модели){
this.имя_коллекции = имя_модели;
        this._model = мангуст.модель(this.имя_коллекции, новая схема(schema));
}
```


Листинг 12.
метод “_initiate” основного класса модели

Список
12 показана реализация метода “_initiate” основного класса модели.
метод принимает схему мангуста и имя модели в качестве аргументов. Первая строка присваивает имя модели для переменной класса с именем “имя_коллекции”. Вторая и последняя строка
отвечает за запуск модели мангуста с использованием предоставленных схемы и модели
имя. Инициированная модель затем сохраняется в th
е свойство класса “_model”. Про-
видимая схема может быть определена в двух местах. Прежде всего, если модель не требует никаких конкретные методы, то схема должна быть определена в файле “Schemas”, который находится в каталоге “system”. И в этом случае нет необходимости создавать отдельный класс для
модель. С другой стороны, если модель действительно требует некоторых методов, которые являются исключительно
используется с этой моделью, она должна быть определена как отдельный класс, расширяющий “AppModel”
класс. Если модель подпадает под вторую категорию, то схема должна быть определена в

свойство класса “schema”. Схему для одной из моделей можно увидеть следующим образом:
низкий:

```
{
"имя пользователя": Строка,
"пароль": Строка,
"contact_info" : {
"телефон" : {тип : Строка, по умолчанию: ""},
"электронная почта": {тип: Строка, по умолчанию: ""},
"фамилия" : {тип : Строка, по умолчанию: ""},
"имя" : {тип : Строка, по умолчанию: ""}
},
"business_info" : {
"тип" : {тип : Строка, по умолчанию: ""},
"адрес": {тип : Строка, по умолчанию: ""},
"число" : {тип : Строка, по умолчанию: ""},
"orgname" : {тип : Строка, по умолчанию : ""}
},
"роль" : Строка,
"одобрено" : {тип : логическое значение, по умолчанию : false},
"финансы": {тип : Объект},
        "активный" : {тип : логическое значение, по умолчанию : false}
}
```

Листинг 13. Схема для пользовательской модели

Схема, показанная в листинге 13, относится к пользовательской модели. Он описывает каждое поле, его тип
и значение по умолчанию. Прежде всего, у пользователя есть имя пользователя и пароль. Во-вторых, пользователь
имеет такую роль, как администратор, организация или клиент. Затем пользователь может быть одобрен и
активирован, поэтому требуются статусы одобрен и активен. И, наконец, есть пер-
личная информация: контактная, деловая и платежная информация, которая управляется в кабинете контроллер.

4.3.4 Просмотры

Представления в проекте создаются с использованием механизма шаблонов EJS. Существует несколько типов
видов в приложении, описанном в этой статье. Прежде всего, общие представления, такие как
“layout.ejs”, “body.ejs”, “javascript_global.ejs” и т.д. Во-вторых, взгляды, связанные с
конкретный контроллер и он

метод s, например, в каталоге под названием “кабинет” есть

представление под названием “info.ejs”, которое является представлением по умолчанию для метода

“info” в разделе "кабинет".

троллер. И, наконец, существуют отдельные элементы, которые используются в представлениях, связанных с
контроллеры.

Представления имеют следующую общую структуру и поток. Некоторый контроллер отображает пар-
вид сверху. Каждое представление определяет макет, с которым должно использоваться представление, используя
следуя синтаксису EJS:

```
<% макет('./__макет/layout') -%>
```

Листинг 13. Определение макета EJS

В листинге 13 показано

код для определения макета представления. Файл “layout.ejs”, описанный

выше, включает в себя “styles_global.ejs”, который содержит все CSS, используемые в глобальном

приложении. В зависимости от статуса аутентификации файл “layout.ejs” также включает либо “body.ejs”, либо

“body_no_auth.ejs”. Оба представления, в свою очередь, включают файл “javascript_global.ejs”, который определяет все библиотеки JavaScript, используемые в проекте, такие как “jquery”, “toastr”, “bootstrap” и т.д. И представление “body.ejs” также включает в себя “left_menu.ejs”, которое создает меню навигации.

Большинство строк в представлениях заняты необработанной HTML-разметкой, и только часть имеет EJS синтаксис для отображения некоторых динамических данных. Пример кода для небольшой части поддержки
просмотр чата можно увидеть ниже:


```
<класс div="список пользователей">
<% для (пусть i = субъекты.длина-1; i >= 0; i--){ %>
<div class="тема чата-пользователя" data-subject="<%= темы [i] %>">
<класс div="чат-имя пользователя">
<%= субъекты[i] %>
</div>
</div>
        <% } %>
</div>
```

Листинг 14. Пример кода EJS для списка разговоров

В листинге 14 показан способ вставки данных в HTML с использованием синтаксиса EJS. Код пример, показанный в листинге 14, создает список всех разговоров в чате поддержки. Как можно увиденный код EJS завернут в
теги “<% %>”, которые называются “скриплеты”. Внутри есть
повторите темы разговора. Весь HTML-код, который находится внутри цикла, будет создан для каждой итерации. Внутри цикла значение каждой темы разговора вставляется в
два места , использующие
теги “<%= %>”, которые выводят значение в шаблон. Таким же образом
весь динамический контент создается в проекте.

5

Результаты

Результатом этой диссертации стало работающее веб-приложение со списком функций. Хотя, это не готовый продукт, он имеет базовую структуру и ряд рабочих частей и
компоненты. В то же время, поскольку основной фреймворк полностью функционален, больше возможностей
и функциональность может быть легко добавлена в приложение. Следующие скриншоты продемонстрировать несколько частей приложения:

Рисунок 11. Интерфейс администратора для управления учетными записями

Рисунок 11 иллюстрирует интерфейс администратора для управления учетными записями для различных организации. Таблица содержит основную информацию об организациях, и это
возможность деактивации и повторной активации учетных записей в любое время.

Рисунок 12. Интерфейс чата поддержки

На рисунке 12 показан интерфейс чата поддержки. Этот модуль доступен администраторам и организации. На этой странице есть список сообщений выбранного разговора
с определенной организацией. Дата каждого сообщения и, в частности, последнего в
текущий разговор можно увидеть на рисунке. Список всех доступных разговоров
с соответствующими заголовками и названием организации, инициировавшей обсуждение, является показано справа от рисунка.

40 (45)
Рисунок 13. Интерфейс для создания нового турнира
На рисунке 13 показан интерфейс организации для создания нового турнира. Как можно видеть
из рисунка можно указать основную информацию о турнире, такую как название,
место и даты проведения турнира. Различные категории с различными настройками могут быть
также добавлено. Также можно указать поля, которые обязательны для заполнения всеми участниками
41 (45)
заполнить при регистрации. Варианты оплаты также доступны для выбора. И
после сохранения созданного турнира он будет доступен для регистрации в интерфейсе участника.
Основная цель статьи по разработке работающего веб-приложения была успешно выполнена
достигнут. Технологии, использованные в проекте, были тщательно изучены и описаны.
Были внедрены хорошая структура и организация кода. Базовая структура с простым,
был разработан понятный поток приложений и пространство для будущих улучшений.
42 (45)

6

Обсуждение

Приложение было разработано с использованием MongoDB и Node.js . Рендеринг на стороне сервера с
для создания пользовательских интерфейсов использовались необработанные CSS и JS. Код был разделен на различные
различные модули, такие как представление, модели и контроллеры. Следовательно, оставляя код чистым
и понятный, и очень простой в обслуживании и расширении в будущем.
Однако, как и любой проект, разрабатываемый продукт он не идеален. В приложении отсутствуют
автоматическое тестирование, такое как модульное, приемное и интерфейсное тестирование со скриншотами. Из
конечно, приложение постоянно тестировалось вручную, но это не устраняет все
ошибки и баги. Следовательно, некоторые неожиданные ошибки или поведение могут возникать во время
с помощью приложения.
В проекте нет интерфейсных фреймворков, таких как React, Angular или Vue.js . Вкл
с одной стороны, поскольку проект, описанный в документе, не является одностраничным приложением (SPA),
у него нет длительного времени начальной загрузки. Каждая страница запрашивается с сервера в
отдельный запрос, и, следовательно, нет необходимости загружать все ресурсы сразу. Вкл
с другой стороны, СПА-это очень популярный подход в наши дни, и у него есть много преимуществ.
Прежде всего, можно перенести маршрутизацию с сервера на интерфейс и использовать сервер
только как источник данных. Затем очень легко модулировать компоненты с помощью
интерфейсных фреймворков. В то же время было бы неплохо отойти от
простые HTTP-запросы CRUD и используют такие современные технологии, как GraphQL с
помощь клиента Apollo во внешнем интерфейсе. GraphQL значительно упрощает запрос только
необходимые данные с одной и той же конечной точки без необходимости создавать десятки конечных точек HTTP
баллы за каждый вид данных и потребностей.
Следовательно, несмотря на то, что проект получился чистым, организованным и функциональным
прототип, всегда существует множество различных технологий и подходов, которые могли бы быть
используется, а также очень обширное пространство для улучшений.
43 (45)

7

Заключение

В центре внимания этой диссертации было изучение технологий, задействованных в полнотекстовых JavaS-
приостановите разработку и создайте работающее веб-приложение на основе изученных тем.
исследования заняли большое количество времени, но в результате были созданы все необходимые

технологии, такие
как Node.js , MongoDB, Express были тщательно проанализированы. Их преимущества и недостатки-
были выявлены преимущества и проведено сравнение с другими технологиями, которые также
используются в
веб-разработка.
После детального изучения теоретического аспекта проекта, практическая часть была
выполнено. Тщательный дизайн архитектуры, а затем фактическая реализация приложения
в результате появился рабочий прототип платформы для управления танцевальными турнирами.
Целью разработки было показать реализацию изученных тем
и удачное сочетание отдельно работающих деталей. Проект может быть продолжен
улучшен рядом функций и преобразован в готовую к производству систему.
После детального исследования и фактической реализации проекта с использованием Node.js и
MongoDB, можно сделать несколько выводов. Прежде всего, это требует
знание только одного языка программирования, что может быть необходимо для небольших компаний
nies, поскольку это позволяет нанять только одного разработчика. Во-вторых, это очень масштабируемая
среда
это может быть использовано для быстрой разработки и быстрой настройки веб-сервера. Однако
технологии могут быть также применены для высоконагруженных проектов с помощью MongoDB shar-
звон, репликация и Node.js кластеризация. В-третьих, существует огромное сообщество JavaScript
это может оказать очень большую помощь. И, наконец, можно сделать вывод, что Node.js , Пн-
Пакет goDB и Express bundle - один из лучших вариантов для разработки современных веб-приложений
развитие.

Ссылки

1
Кодесидо I. Что такое интерфейсная разработка? [онлайн]. Официальный сайт Guardian; 28
сентября 2009
года
URL-АДРЕС:
https://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost
.
Дата обращения: 6 ноября 2016 года
2
Консорциум Всемирной паутины. Что такое CSS? [онлайн]. Официальный сайт W3.
URL-АДРЕС:
https://www.w3.org/standards/webdesign/htmlcss#whatcss
.
Дата обращения: 6 ноября 2016 года
3
Стивен М. Шафер. Библия HTML, XHTML и CSS, 5-е издание. Crosspoint Буль-
вард, В: Wiley Publishing; 2010
4
JavaScript
– Обзор [онлайн]. Официальный сайт TutorialsPoint.
URL-АДРЕС:
https://www.tutorialspoint.com/javascript/javascript_overview.htm
.
Дата обращения: 6 ноября 2016 года
5
Что такое веб-сервер? [онлайн]. Официальный сайт MDN.
URL:
https://developer.mozilla.org/en-US/docs/Learn/Common_ques -

связи/Что_ Это_a_web_server
.
Дата обращения: 6 ноября 2016 года
6
Опрос веб-сервера в сентябре 2016 года [онлайн]. Официальный сайт Netcraft; 19 сентября-тембр 2016
URL-АДРЕС:
https://news.netcraft.com/archives/2016/09/19/september-2016-web -
server-survey.html
Дата обращения: 6 ноября 2016 года
7
Серверные веб-фреймворки [онлайн]. Официальный сайт MSDN.
URL-АДРЕС:
https://developer.mozilla.org/en-US/docs/Learn/Server -
сторона/Первые шаги/Web_frameworks
Дата обращения: 12 февраля 2017 года
8
Руководство по бизнес-уровню [онлайн]. Официальный сайт MSDN.
URL-АДРЕС:
https://msdn.microsoft.com/en-in/library/ee658103.aspx
Дата обращения: 11 февраля 2017 года
9
NoSQL против SQL [онлайн]. Официальный сайт MSDN.
URL-АДРЕС:
https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs -
sql
Дата обращения: 12 февраля 2017 года
10
Ливитт Н. Оправдают ли базы данных NoSQL Свои обещания? Компьютер. Февраль 2010, 43(2).
11
Node.js официальный сайт [онлайн].
URL-АДРЕС:
https://nodejs.org
Дата обращения: 15 апреля 2017 года
12
Красимир Цонев. Node.js На примере. Бирмингем, Великобритания: Packt Паб-
срок действия ограничен;
25 Мая 2015
года
45 (45)
13
Синко де НодеЙС
— Майский BayJax празднует серверный JavaScript с Райаном
Даль, Элайджа Инсуа и Дав Гласс. ЮИ; 27 апреля 2010
14
Node.js официальный веб-сайт API [онлайн].
URL-АДРЕС:
https://nodejs.org/api/cluster.html
Дата обращения: 14 мая 2017 года
15

Кай Лэй, Инин Ма, Чжи Тан. Сравнение производительности и оценка технологий веб разработки на PHP, Python и Node.js . Вычислительная наука и инженерия (CSE), 2014 17-я международная конференция IEEE по; 2014
16
Официальный сайт НПМ [онлайн].
URL-АДРЕС:
https://www.npmjs.com
Дата обращения: 14 мая 2017 года
17
Саймон Холмс. Получаем СРЕДНЕЕ значение с помощью Mongo, Express, Angular и Node. Публикации Мэннинга; 2015
18
Официальный сайт Express [онлайн].
URL-АДРЕС:
http://expressjs.com
Дата обращения: 21 мая 2017
19
Официальный сайт MongoDB [онлайн].
URL: https://www.mongodb.com
Дата обращения: 15 июля 2017 года
20
Официальная документация MongoDB [онлайн].
URL: https://docs.mongodb.com /
Дата обращения: 16 июля 2017
21
Официальный сайт Мангуста [онлайн].
URL: http://mongoosejs.com /
Дата обращения: 21 июля 2017