Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Разработка веб-приложения для управления контентом мобильной среды для изучения интонации.

Студент гр. 43501/3 М.С. Мальцев

Санкт-Петербург
2019

Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Работа допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«___» _____ 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Разработка веб-приложения для управления контентом мобильной среды для изучения интонации.

по направлению 09.03.01 «Информатика и вычислительная техника»
по образовательной программе
09.03.01_02 «Технологии разработки программного обеспечения»

Выполнил студент гр. 43501/3 _____ М.С. Мальцев

Научный руководитель,
к. т. н., доц. _____ Н.В. Богач

Санкт-Петербург
2019

РЕФЕРАТ

На 58 с., 9 рисунков, 1 приложение

ОДНОСТРАНИЧНОЕ ВЕБ-ПРИЛОЖЕНИЕ, СТЕК ТЕХНОЛОГИЙ MERN, STUDYINTONATION

В работе рассматривается проектирование и разработка веб-приложения для управления контентом мобильной среды для изучения интонации. Изучение интонации - важная часть изучения языка, но на сегодняшний день не существует законченных свободно распространяемых продуктов, которые позволяли бы их изучать. Разрабатываемое веб-приложение является частью проекта “Study Intonation”, который ставит целью создать открытую среду для изучения интонаций.

Мотивация разработки приложения - предоставить пользователям проекта “Study Intonation” возможность создавать, редактировать и распространять курсы в удобной форме. Также в мотивацию включена цель изучить используемый технологический стек и полный процесс создания одностраничного веб-приложения.

Работа сконцентрирована вокруг проектирования и реализации веб-приложения. При проектировании выделено 5 компонентов: клиентская часть, серверная часть, база данных, упаковщик и файловое хранилище. В качестве технологического стека выбран MERN + AWS S3. Разработанные компоненты размещены в Docker-контейнерах.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Благодаря появлению и развитию современных информационно-коммуникационных технологий сформировалась совершенно новая область в сфере образования - электронное обучение. Такая точка зрения подкрепляется работами [6, 9], в которых представлены рассуждения о будущем образования и эффективном обучении.

В работах [1, 19, 20, 23], которые освещают аспекты дистанционного обучения, в качестве ключевых преимуществ электронного обучения для слушателей выделяется: гибкость, доступность и самоорганизованность. А для преподавателей: низкая себестоимость и возможность привлечения большой аудитории.

Электронное обучение, несет в себе не только количественное улучшение существующей системы, но и качественное. Например, в работе [24], посвященной информационным технологиям в обучении иностранному языку, утверждается: “Под применением новых информационных технологий в обучении иностранным языкам понимают не только использование современных технических средств и технологий, но и использование новых форм и методов преподавания иностранного языка и новый подход к процессу обучения в целом.”.

Таким образом, определяется вектор изменений, который несёт в себе многогранное улучшение образовательной системы.

В качестве проекта, который не просто переводит процесс изучения языков в интернет пространство, а предлагает новый способ освоения языка, через изучение интонации, выступает “Study Intonation” - мобильная среда для изучения интонаций [13]. Проект состоит из двух частей: мобильное приложение для работы с курсами и веб-приложение для их создания. В задачи мобильного приложения входит загрузка курсов и обеспечение взаимодействия

с пользователем: запись речи пользователя, вычисление метрики соответствия между интонацией обучающегося и эталонным значением, которое заложено в курсе, и вывод результатов для корректировки произношения.

В данной работе акцент сделан на разработке веб-приложения для создания, редактирования и распространения курсов. Работа сконцентрирована вокруг проектирования и реализации веб-приложения.

Мотивация разработки приложения - предоставить пользователям проекта “Study Intonation” возможность создавать, редактировать и распространять курсы в удобной форме. Также в мотивацию включена цель изучить используемый технологический стек и полный процесс создания одностраничного веб-приложения.

Для создания веб-приложения используется стек MERN (MongoDB, Express, React, Node.js). MongoDB - документоориентированная СУБД. Node.js - программная платформа позволяющая запускать JavaScript вне браузера. Express - минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. React - JavaScript-библиотека для создания пользовательского интерфейса. Для размещения медиафайлов используется файловое хранилище реализующее интерфейс AWS S3. Разработанные компоненты размещены в Docker-контейнерах и запускаются с помощью Docker Compose.

1. АНАЛИЗ

В главе описано позиционирование разрабатываемого продукта, необходимость применения собственного решения, а также функциональные и нефункциональные требования, предъявляемые к разрабатываемому продукту.

1.1. Обзор предметной области

В современном интернет пространстве большая часть контента, создается пользователями. Именно на этом принципе построена парадигма Web 2.0. Термин Web 2.0 описывает второе поколение WWW, которое отличается от первого поколения более высокой интерактивностью и динамичностью, а также фокусируется на пользователях и их способностях объединяться в сообщества и делиться информацией [18]. С рассматриваемым термином связаны такие понятия, как обмен, коллаборации, социализация, однако, ключевым преимуществом Web 2.0 является возможность использовать коллективный интеллект, для формирования контента, наполняющего интернет пространство. Хотя изначально Web 2.0 не был предназначен для образовательных целей, он отлично справляется с задачами личностного и профессионального развития. Например, в работе [5] отмечается высокий потенциал в использовании технологий Web 2.0 для образовательной сферы.

Если же рассматривать более узкую область применения рассматриваемых технологий, а именно, изучение иностранных языков, то и в данном случае, Web 2.0 хорошо себя зарекомендовал. В работе [3] говорится, что преподаватели иностранных языков особенно заинтересованы в новых технологиях, которые позволят учащимся улучшить их языковые навыки.

Таким образом, идея использования принципов Web 2.0 в проекте “Study Intonation” полностью обоснована.

Важным остается место педагога в новой среде. В работе [12] в качестве одной из основных задач, которые остаются за преподавателем, выделяется создание онлайн курсов. Но существуют сложности связанные с тем, что не все учителя умеют пользоваться новыми технологиями, о них упоминается в статьях [8,16]. Преподавателям необходимо адаптироваться к новому интернет поколению, которое с детства обладает навыками работы с компьютерами, и корректировать свои методики обучения для более эффективного их применения [11].

Для того, чтобы помочь преподавателям сориентироваться в современных технологиях и способах их применения были организованы такие проекты, как “SLOOP”, “Tenegen” [1], позже “Sloop2desc” [2], в России - проект “ШЛЮПА”. Идея проектов заключалась в том, чтобы “применить философию бесплатного программного обеспечения к производству педагогических материалов для электронного обучения” [22]. Но, к сожалению, все перечисленные выше проекты прекратили свое существование. “Study Intonation” поддерживает подобную философию и основывается на похожих принципах.

При рассмотрении современных средств для изучения иностранных языков, находящихся в свободном доступе, следует отметить что большая часть из них нацелена на расширение словарного запаса и объяснение грамматических основ языка, а сами системы в своем большинстве закрыты для добавления обучающего контента, создаваемого пользователями. Если же такая функция имеется, то обычно она является платной. Подобный подход, безусловно, является коммерчески обоснованным, но, таким образом, он нарушает

[1] http://tenegen.eu/
[2] http://www.sloop2desc.eu/

идею свободного электронного обучения.

Интонации играют важную роль в разговорной речи человека. Например, в работе [17], посвященной проблеме преподавания интонаций, говорится о том, что в некоторых случаях интонации несут больше информации, чем слова. Однако на изучение интонаций в преподавании языков отводится не так много времени. О важности изучения интонаций пишут в статье [15]. В ней автор рассуждает над тем, стоит ли вообще преподавать интонации, и приходит к выводу, что для более успешного освоения языка интонации необходимы.

На данный момент не существует свободно распространяемого законченного продукта, который бы предлагал изучать иностранные языки делая при этом акцент на интонациях, а следовательно и ни одного ресурса, который бы позволял создавать курсы для подобной задачи. В статье [10] приводится несколько приложений, который ставят перед собой похожие задачи, но они обладают скудной функциональностью и распространяются платно. Именно подобная проблема ставит задачу разработки собственного решения. Кроме того, стоит отметить, что мобильное приложение для изучения иностранных языков уже создано и на момент написания работы находится на этапе бета-тестирования, а, следовательно, формат задания курсов уже определен

Структура курсов состоит из JSON-файла, медиафайлов и файлов с отсчетами питча. В JSON-файле содержится описание курса, а также уроков и задач, из которых состоит курс, также в нём присутствуют ссылки на файлы, которые используются в курсе.

Несколько уже созданных курсов формировались вручную, т.е. пользователь, создающий контент, с помощью примитивного текстового редактора формировал JSON-файл и по указанным в нём

ссылкам размещал необходимые файлы. Подобный способ формирования контента является не удобным для пользователей по ряду причин:

1. ошибки, допущенные при формировании, курса сложно отслеживать и исправлять
2. формат используемого JSON-файла понятен не всем пользователям

Исходя из этих предпосылок было решено разработать сервис, который позволит в удобном формате создавать, редактировать и распространять контент для мобильной среды.

1.2. Функциональные требования к архитектуре

Необходимо создать систему, которая позволяет создать, редактировать и распространять курсы для изучения языков с акцентом на интонациях. Система должна представлять графический интерфейс и обеспечивать доступ к нему с помощью браузера. Пользователь использующий приложение, должен иметь доступ к уже существующим курсам. Возможность зарегистрироваться, если же он ранее был зарегистрирован, то войти в систему. После аутентификации, должна открываться возможность создать новый курс, а также редактировать курсы ранее добавленные этим пользователем.

При создании и редактировании курса должна быть реализована возможность ввода названия, описания, уровня сложности, категории, и указания авторов, а также загрузки изображения в форматах *.jpg и *.png для обложки курса.

При создании урока или его редактирования пользователю должны быть доступны поля: название, описание, номер, длитель-

ность. При создании и редактировании заданий необходимо обеспечить возможность заполнения полей: номер, инструкции пользователю и текст задания. Должна быть реализована возможность загрузки аудиофайлов в форматах *.wav и *.mp3, а также вычисления по аудиофайлу питча и сохранения его в отдельном файле с расширением *.pitch.

Также по запросу пользователям система должна организовывать архивирование JSON-файла и медиафайлов, относящихся к выбранному курсу, с последующим размещением его для дальнейшего скачивания. При этом должен быть доступен для скачивания JSON-файл, который содержит краткое описание курсов для уведомления мобильного клиента о новых и обновленных курсах.

1.3. Нефункциональные требования к архитектуре

Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям:

- Приложение должно обеспечивать правильное отображение сайта на различных устройствах, подключенных к интернету, и динамически подстраивающийся под заданные размеры окна браузера
- Разрабатываемый графический интерфейс должен демонстрировать высокие пользовательские качества
- Продукт должен легко масштабироваться без серьезных изменений в коде
- Приложение должно легко запускаться и разворачиваться, на не специализированом хостинге

1.4. Результаты

В результате анализа предметной области сделан вывод, что при изучении иностранных языков подход основанный на интонациях обоснован. Аналогов для проекта “Study Intonation” почти нет, а те что есть предоставляют скудную функциональность и распространяются платно. Как следствие, сервисы, позволяющие создавать курсы подобного типа, тоже отсутствуют. А также в условиях уже существующего мобильного приложения, необходима разработка своего собственного продукта, который будет позволять создавать, редактировать и распространять контент для мобильной среды. Описаны функциональные и нефункциональные требования к разрабатываемому веб-приложению.

2. ПРОЕКТИРОВАНИЕ

В главе описано проектирование архитектуры приложения. Определены взаимоотношения между компонентами и технологический стек.

2.1. Архитектура

Архитектура разрабатываемого приложения состоит из 5 частей:

- база данных
- сервер
- клиент
- файловое хранилище
- упаковщик

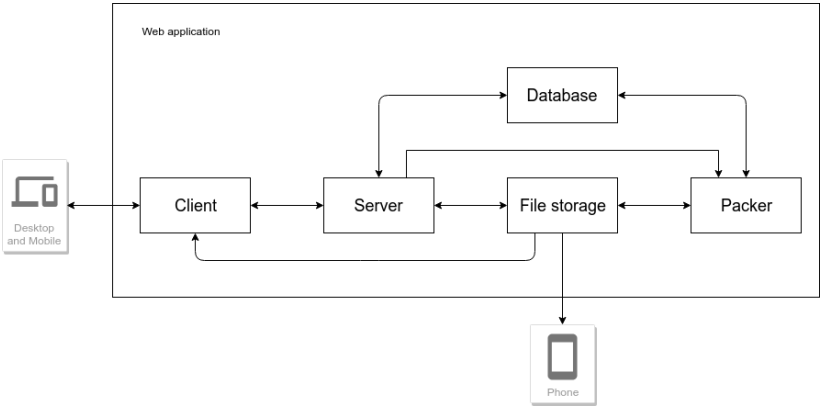


Рис.2.1. Архитектура разрабатываемого приложения. Компоненты и взаимоотношения между ними.

К преимуществам подобного разделения относится легкая масштабируемость системы, так как при увеличении нагрузки на один из компонентов его можно заменить другим, более производительным. Например, при увеличении количества медиафайлов, размещающихся на файловом хранилище, можно сменить его, купив больший объем пространства, при этом остальные компоненты системы останутся неизменными. Причем подобные операции можно совершать над любым из компонентов. В качестве преимущества используемого подхода стоит упомянуть надежность системы. В случае отказа серверной части приложения, файловое хранилище все еще будет доступно, и пользователи смогут скачать курсы, необходимые для работы мобильного приложения. В следующих подпунктах каждый из компонентов, его назначение и функции описаны отдельно.

2.1.1. База данных

База данных хранит информацию о пользователях и курсах. Первоначально планируется небольшое количество пользователей, примерно от 100 до 1000. Однако структура курсов предполагается довольно объёмной. В один курс может входить более 15 уроков и в каждом уроке около 20 заданий. Учитывая, что у пользователей может быть по несколько созданных курсов, объем данных предполагается средний. Но если учесть, что при поддержке приложения структура курса может измениться, наиболее удачным выбором системы управления базами данных будет NoSQL решение. Подобные базы данных не имеют четкой структуры, что позволит изменять схему курса без вреда уже существующим наработкам.

2.1.2. Серверная часть

Серверная часть веб-приложения решает такие задачи, как получение запросов с клиентской части, их обработка и ответ, заключающийся в отправке клиенту запрашиваемой информации в JSON формате. Сервер имеет доступ к базе данных и файловому хранилищу.

В данном случае не требуется какой-то сложной функциональности со стороны сервера, помимо вычисления питча. Но и эту задачу на себя берёт сторонняя библиотека на C++. Следовательно, наиболее релевантным в данном случае будет выбор технологии, которая позволит максимально просто создать веб-сервер, который будет легко взаимодействовать с NoSQL базой данных, файловым хранилищем и без особых сложностей сможет вызвать нативный код, написанный на C++.

2.1.3. Клиентская часть

В задачи решаемые клиентской частью веб-приложения входит представление пользователю графического интерфейса для визуализации информации приходящей с серверной части веб-приложения, фиксация действий пользователя и уведомление о них серверной части. Стоит отметить, что клиентская часть веб-приложения выполняется в веб-браузере пользователя. Так как предполагается, что пользователи будут создавать и редактировать курсы, уроки и задания, то ожидаемо частое обновления контента на веб-странице, как следствие, наиболее оптимальным решением будет выбрать одностраничную архитектуру приложения, так как она позволяет обновлять не всю веб-страницу целиком, а лишь изменившиеся ее части.

2.1.4. Файловое хранилище

Файловое хранилище необходимо для размещения курсов пользователей. Использование хранилища обеспечивает надежность веб-приложения, в случае если сервер откажет, так как доступ к скачиванию курсов на мобильное устройство всё ещё будет открыт. Предполагается использовать сервис реализующий Amazon S3 протокол, так как в случае критичного увеличения объема данных, которые необходимо хранить, появляется возможность перенести файловое хранилище на сервера Amazon, тем самым решив проблему масштабирования для этого компонента.

2.1.5. Упаковщик

Этот компонент представляет из себя сервер который ждёт запрос с идентификатором курса. Имеет доступ к базе данных и файловому хранилищу. После получения запроса извлекает из базы данных информацию о запрашиваемом курсе, а из файлового хранилища медиафайлы, принадлежащие курсу архивирует их и отправляет в файловое хранилище, попутно изменив файл, с информацией о доступных для скачивания мобильному приложению курсах. Эта функциональность никак не связана с функциональностью, которая предоставляется серверной частью, следовательно, может быть выделена в отдельный компонент.

2.2. MERN + AWS S3

MERN [14] это программный стек технологий использующийся для создания динамических веб-приложений. Он составляет комбинацию 4 популярных технологий:

- MongoDB [1] - NoSQL база данных
- Express.js [2] - веб-фреймворк для Node.js
- React.js [3] - JavaScript фреймворк для создания SPA приложений
- Node.js [4] - платформа для выполнения JavaScript

2.2.1. Node.js

Node.js является серверной технологией, которая основана на разработанном компанией Google JavaScript-движке V8 и позволяет выполнять JavaScript сценарии вне браузера.

В качестве преимуществ платформы стоит отметить:

- универсальность (один язык сервера и клиента)
- легкая работа с нативными модулями на C и C++
- низкая требовательность к ресурсам и отсутствие беспокойства относительно программных потоков
- хорошая масштабируемость
- NPM (Node Package Manager), пакетный менеджер, который обеспечивает доступ к множеству различных инструментов и модулей

В качестве особенностей работы Node.js можно выделить то, что для каждого запроса к серверу не создается новый программный поток или процесс, а идёт прослушивание конкретных собы-

[1] https://www.mongodb.com/
[2] https://expressjs.com
[3] https://reactjs.org
[4] https://nodejs.org

тий, и когда эти события происходят, сервер соответствующим образом на них реагирует. При работе Node.js не блокирует никаких запросов, дожидаясь завершения действий, инициируемых событием, а сами события обрабатываются в относительно простом цикле обработки событий по принципу «первым пришел — первым обслужен» [21]. Используемые в Node цикл обработки событий и функции обратного вызова имеют два основных преимущества.

Первое из них состоит в том, что приложение лучше масштабируется, поскольку у одного программного потока не так уж и много накладных расходов.

Второе преимущество Node заключается в минимизации расходования ресурсов, не прибегая для этого к многопоточной разработке, то есть не создавая многопоточное приложение.

Так как в данной работе нет жестких функциональных требований по производительности, а остальным требованиям Node.js удовлетворяет, то эта технология, может быть использована для реализации серверной части приложения.

2.2.2. Express

Express - это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Использование Express предоставляет достаточный набор программных компонентов, чтобы разработка была сконцентрирована на уникальной для приложения или сайта функциональности [7]. Также этот фреймворк позволяет превратить монолитную обработку запросов в множество небольших обработчиков, которые обрабатывают небольшие части, что обеспечивает удобную модульную разработку [4].

Express поддерживает утилиту "express-generator", которая позволяет создать структуру простого веб-приложения, на основе

которого планируется разрабатывать серверную часть.

2.2.3. MongoDB

MongoDB нереляционная, документоориентированная, кроссплатформенная система управления базами данных с открытым исходным кодом. MongoDB предлагает гибкую структуру хранения данных, что удобно в рамках решаемой задачи. Также рассматриваемая СУБД может быть легко масштабируема и не требует предопределенных схем для начала работы.

Для того чтобы связать базу данных с Node.js планируется использовать ODM. Самое популярное решение в данном случае - Mongoose [5]. Mongoose - это JavaScript библиотека, позволяющая определять схемы со строго-типизированными данными, создавать модель, основанную на определенной схеме, и синхронизировать её с документом в MongoDB.

2.2.4. React

React — это популярная JavaScript-библиотека с открытым исходным кодом для создания одностраничных приложений. Благодаря применяемому подходу к проектированию пользовательского интерфейса React позволяет переиспользовать UI-компоненты. Возможности React позволяют разработчикам создавать большие вебприложения, которые позволяют изменять контент страницы без ее перезагрузки [2]. К преимуществам этой библиотеки стоит отнести высокую скорость работы, масштабируемость и простоту.

К особенностям React следует отнести Virtual Document Object Model - абстракция над HTML DOM. Подобное решение позволяет React выполнять вычисления над абстракциями и пропускать

[5] https://mongoosejs.com

реальные операция над DOM, часто довольно медлительные.

Учитывая, что графический интерфейс планируется разрабатывать для сравнительно небольшого количества предоставляемой функциональности: создания и редактирования курсов, уроков и задач, то использовать для этого большие фреймворки неэффективно. В тоже время, ввиду расчета на то, что пользователь будет часто переключаться между страницами, необходимо применить, именно, одностраничную архитектуру. Как следствие получаем, что самый популярный легковесный фреймворк для создания одностраничного приложения - это React.

2.2.5. AWS S3

Amazon Web Services (AWS) [6] - это защищенная облачная платформа, предлагающая вычислительную мощность, хранение баз данных, доставку контента и другую функциональность, помогающую предприятиям масштабироваться и расти [7]. Но также это и интерфейс взаимодействия. В данной работе используется файловое хранилище реализующее этот интерфейс. Модуль, который был выбран из всего набора предоставляемой AWS функциональности называется S3 (Simple Storage Service). Выбор остановился на нем, так как он самый простой и при этом позволяет решить задачу хранения файлов. В качестве реализации интерфейса AWS было выбрано хранилище Zenko CloudServer [8]. Данная реализация отличается тем, что она находится в свободном доступе под лицензией "Apache License", написана на Node.js и распространяется в виде Docker-образа, что удобно встраивает ее в разрабатываемое веб-приложение.

[6] https://aws.amazon.com/

[7] https://blog.usejournal.com/what-is-aws-and-what-can-you-do-with-it-395b585b03c

[8] https://www.zenko.io/cloudserver/

2.3. Docker

Docker [9] - программное обеспечение для автоматизации развертывания и управления приложениями. Одно из ключевых преимуществ использования Docker - это воспроизводимость. Используется специальный Docker-файл, который описывает спецификацию Docker-контейнера. В итоге получаем гарантию того, что все Docker-образы созданные из файла спецификации будут работать одинаково вне зависимости от окружения в котором они запускаются. Кроме того, использовании Docker-контейнеров позволяет обеспечить необходимый уровень изоляции компонентов и тем самым обеспечить безопасность их функционирования. Стоит отметить, что многие программные продукты, как например, Zenko CloudServer, упоминаемый в прошлом подпункте, распространяется в виде Docker-образа, что облегчает их интеграцию в систему.

Основная идея применения Docker-контейнеров в разрабатываемом проекте - это изоляция независимых частей и упрощение развертывания.

Планируется создать по одному контейнеру, под каждый компонент:

- база данных
- сервер
- клиент
- файловое хранилище
- упаковщик

[9] https://www.docker.com/

Для того, чтобы определить зависимости контейнеров между собой и осуществлять запуск используется инструмент Docker Compose [10].

2.4. Результаты

В результате проектирования составлена архитектура приложения и определены взаимоотношения между компонентами. Решено выделить 5 компонентов: база данных, сервер, клиент, файловое хранилище и упаковщик. Определен стек технологий, выбор сделан в пользу MERN + AWS S3. Приведены преимущества и обосновано применение выбранного стека. Запланирована докеризация приложения. Результат проектирования представлен на рисунке 2.2.

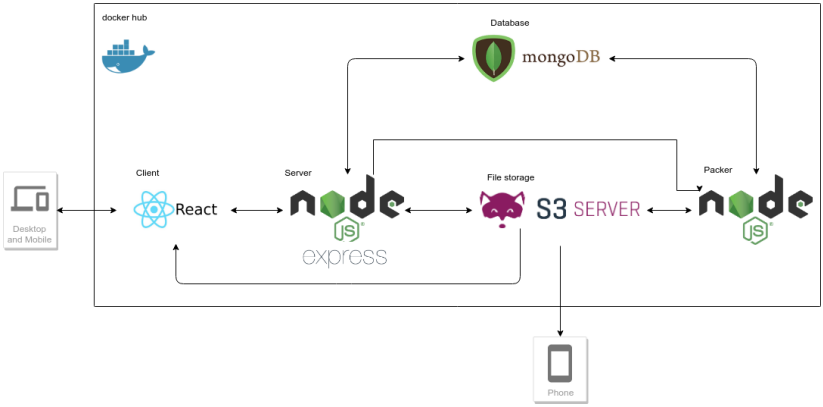


Рис.2.2. Результат проектирования. Архитектура разрабатываемого приложения и используемый стек технологий.

[10] https://docs.docker.com/compose/

3. РАЗРАБОТКА

В этой главе описаны основные зависимости компонентов и принцип их действия. Приведены фрагменты кода и объяснено их действие. Описан процесс и резульаты тестирования для серверной части. Дано описание среды разработки.

3.1. Среда разработки

Используемая операционная система система и прикладные программы:

- OS: Ubuntu 18.04 bionic
- Kernel: x86-64 Linux 4.15.0-50-generic
- IDE: Visual Studio Code
- Browser: Chromium v74.0.3729.108 (Official Build) snap (64-bit) and Firefox Quantum 66.0.5 (64-bit)
- Node.js v10.15.3
- npm v6.4.1
- Docker version 18.06.1-ce
- docker-compose version 1.22.0

3.2. Серверная часть

Для создания серверной части использовались платформа Node.js и фреймворк Express. Сервер обеспечивает взаимодействие клиента с базой данных, файловым хранилищем и упаковщиком.

Первоначальная структура проекта создавалось с помощью утилиты “express-generator”. Ключевой файл серверной части веб-приложение - это app.js. В его задачи входит установка соединений с базой данных, подключение cookie-сессий, и обработка запросов на различные url-адреса. Также в его задачу входит обработка ошибок. Поддерживается обработка ошибки с кодом 404 - “Не найдено” и ошибки с кодом 500 - “Ошибка сервера”.

При получении запроса сервер, в зависимости от того, на какой url запрос отправлен, задает файл обработчик. Например, в случае, когда запрос отправлен на корень сервера, его обработка в app.js будет выглядеть так:

Листинг 3.1. Пример перенаправления обработки запроса на корень сервера для файла app.js

```
const index = require("./routes/index");
       ...
app.use("/", index);
```

Обработка запроса перенаправляется в файл, index.js, который располагается в папке routes. Код расположенный в файле index.js:

Листинг 3.2. Пример обработки запроса на корень сервера для файла index.js

```
const express = require("express"),
      router = express.Router(),
      models = require("../models");

router.get("/", async (req, res) => {
  try {
    const id = req.session.userId;
    const login = req.session.userLogin;

    const courses = await models.Course.find({ published: true }).
        sort({
      createdAt: -1
    });
```

```
    res.json({
      courses,
      user: {
        id,
        login
      }
    });
  } catch (error) {
    throw new Error("Server Error");
  }
});

module.exports = router;
```

Сначала объявляются зависимости, используемые в файле. Далее задается обработчик GET-запроса, при url равному “/index”. Блок try-catch используется для корректной обработки исключений. Далее из запроса извлекаются id и login пользователя, а из базы данных все курсы, с поднятым флагом published, и сортировкой по убыванию для поля даты создания. Собранные данные отправляются клиенту, в JSON формате. Конструкция module.exports, позволяет экспортировать обработчик, чтобы его можно было подключить в файле app.js.

Полный список зависимостей серверной части представлен в файле package.json, но некоторые из них, которые являются ключевыми, описаны ниже:

- aws-sdk - модуль, обеспечивающий доступ к файловому хранилищу, реализующему интерфейс AWS
- mongoose - ODM для MongoDB
- bindings и node-gyp - два модуля, позволяющие вызывать из JavaScript код на C++
- dotenv - модуль, позволяющий задавать параметры сервера, через переменные среды

- bcrypt-nodejs - библиотека позволяющая шифровать пароли
- cookie-session - позволяет использовать cookie для хранения пользовательских сессий
- multer - модуль, использующийся для обработки загрузки файлов

Также у серверной части приложения есть зависимости, которые необходимы лишь в процессе разработки:

- mocha - фреймворк, позволяющий в асинхронном режиме запускать тесты
- chai и chai-http - BDD / TDD библиотека с функциями для проверки внутреннего состояния программы, а также модуль к этой библиотеке, который позволяет в удобной форме отправлять http-запросы
- nodemon - утилита, контролирующая изменения в коде и автоматически перезагружающая приложение
- eslint - инструмент для статического анализа кода

Используемые версии всех подключаемых модулей зафиксированы в файле package-lock.json.

3.3. База данных

В качестве СУБД используется MongoDB. Основная задача базы данных - это хранения информации о курсах и пользователях. Для информации хранения в базе данных используется 5 схем:

- курсы

- уроки
- задачи
- пользователи
- опубликованные курсы.

Курсы, уроки и задачи хранят данные характеризующие контент создаваемый для мобильной среды. Схема с пользователями используется для хранения пароля в зашифрованном виде и логина. Схема опубликованные курсы, хранит краткое описание курсов, ссылки, по которым можно скачать архив курса и дату публикации. Схемами курсы, уроки, задачи и пользователи пользуется для чтения и редактирования серверная часть. Упаковщик пользуется курсами, уроками, задачами для чтения, а опубликованными курсами для чтения и записи.

3.4. Клиентская часть

Для создания клиентской части использовался язык JavaScript и фреймворк React. Компонент, получающий данные в JSON виде с серверной части, обеспечивающий формирование пользовательского интерфейса и отображение его в браузере пользователя. Приложение запускается на отдельном сервере, который проксирует запросы на файловое хранилище и основной сервер. Клиентская часть создана на одностраничной архитектуре.

Для того, чтобы обеспечить обновление контента без перезагрузки страницы используется React компонент BrowserRouter. Он позволяет посылать различные url запросы, не перезагружая страницу. Ниже представлен код, позволяющие в зависимости от url запроса демонстрировать либо страницу для отображения курса, либо страницу для его редактирования.

Листинг 3.3. Пример кода компонента Клиент, демонстрирующий механизм роутинга

```
<Switch>
  <Route exact path="/content/course/:courseId" render={(props) => (
    <Paper className={classes.paper}>
      <ContentCourse {...props} accId={this.props.accId} />
    </Paper>
  )} />

  <Route exact path="/content/edit/course/:courseId" render={(props)
       => (
    <Paper className={classes.paper}>
      <EditCourse {...props} accId={this.props.accId} />
    </Paper>
  )} />
      ...
</Switch>
```

Для создания графического интерфейса, помимо основополагающего фреймворка React, используется React-библиотека Material-UI[1], которая предоставляет набор визуальных решений, соответствующих принципам Material Design. Для визуализации аудиофайла использовалась библиотека wavesurfer.js[2], а для визуализации файла с питчом библиотека dygraphs[3].

На рисунках 3.1, 3.2, 3.3, 3.4 представлен разработанный графический интерфейс приложения.

[1] https://material-ui.com/
[2] https://wavesurfer-js.org/
[3] http://dygraphs.com/

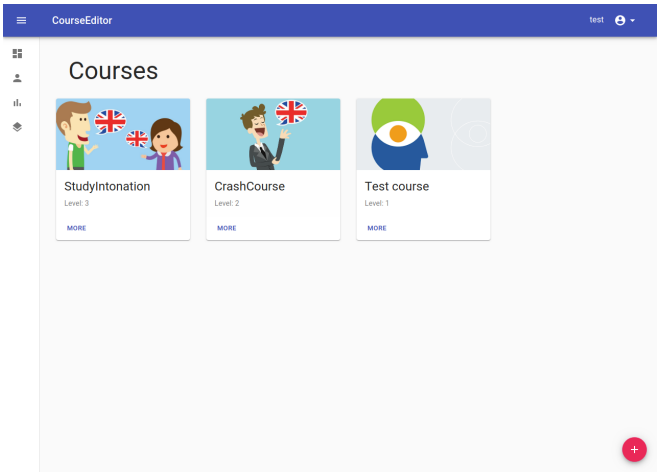


Рис.3.1. Графический интерфейс приложения. Desktop версия. Экран с курсами.

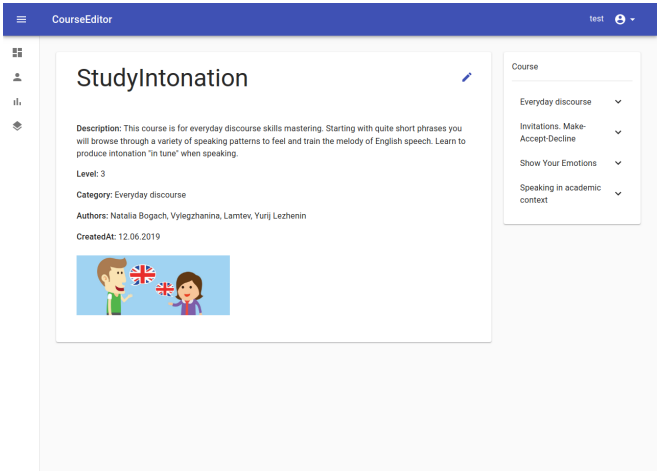


Рис.3.2. Графический интерфейс приложения. Desktop версия. Экран с курсом.

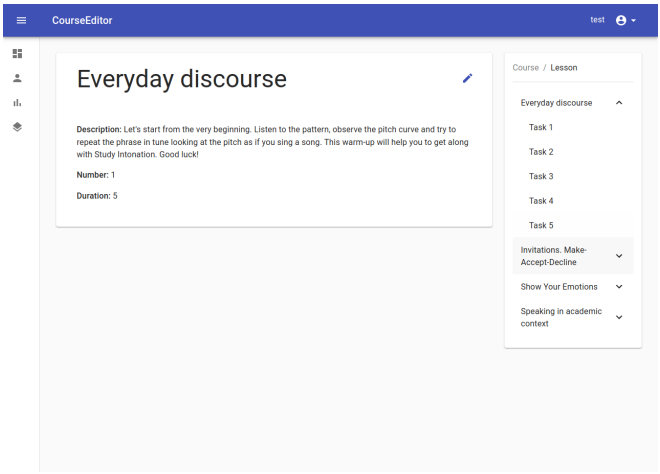


Рис.3.3. Графический интерфейс приложения. Desktop версия. Экран с уроком.

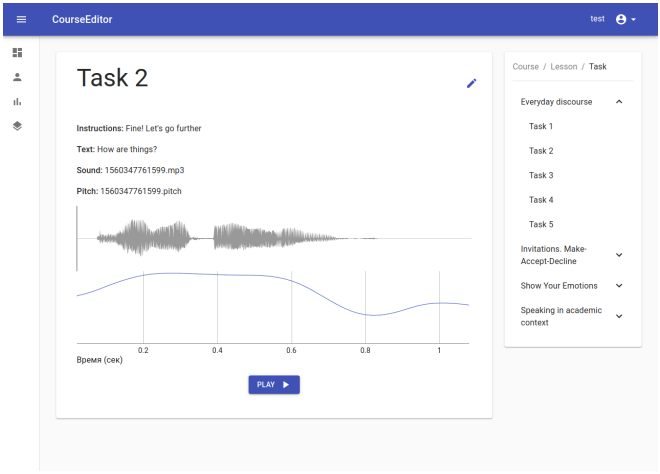


Рис.3.4. Графический интерфейс приложения. Desktop версия. Экран с задачей.

Предусмотрена валидация полей при редактировании и создании контента. Для информирования пользователей о некорректном вводе используются всплывающие уведомления и подсветка форм, в которых допущена ошибка. На рисунке 3.5 продемонстрирована ошибка - незаполненное поле.

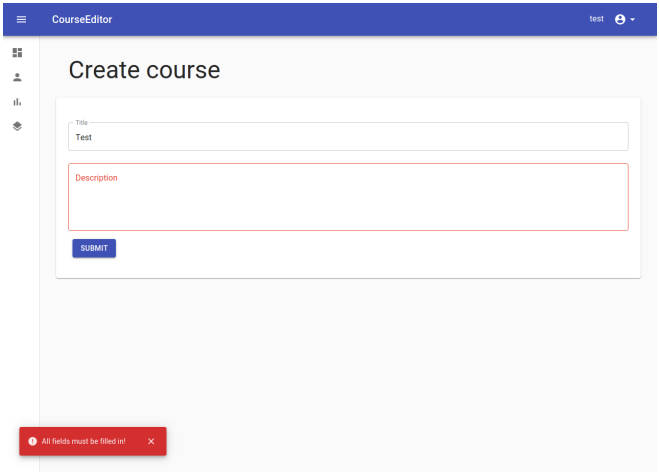
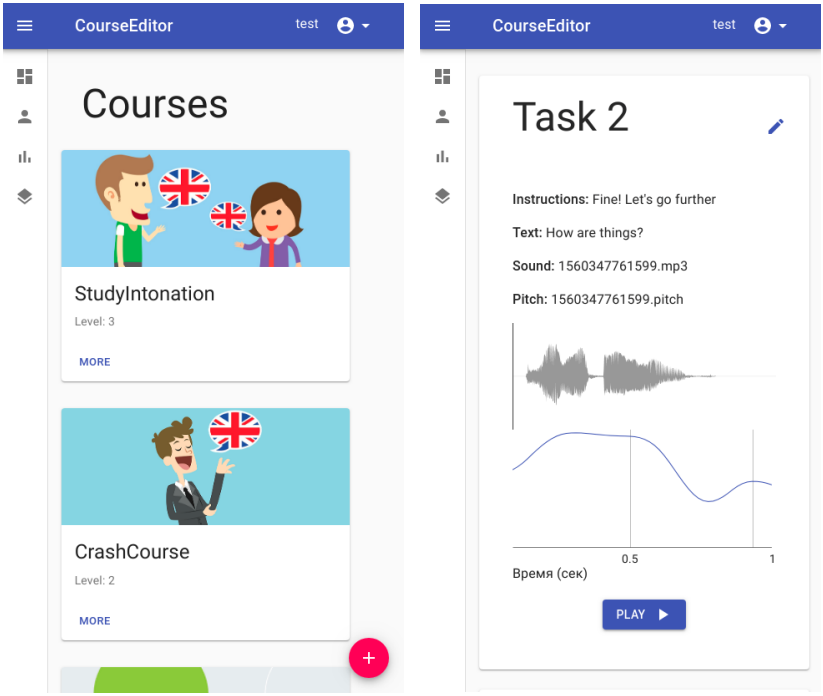


Рис.3.5. Графический интерфейс приложения. Desktop версия. Экран с уведомлением о некорректном заполнении формы.

Разработанное веб-приложение является адаптивным и правильно отображается на устройствах, с разными размерами экрана. На рисунке 3.6 продемонстрировано отображение мобильной версии веб-приложения.



а) Экран с курсами б) Экран с задачей

Рис.3.6. Графический интерфейс приложения. Мобильная версия.

На стороне клиента организована аутентификация и авторизация. Приложение не допускает пользователя до экрана создания или редактирования курса, пока он пройдет не аутентификацию.

3.5. Упаковщик

Для реализации этого компонента используется Node.js. Эта часть приложения, отвечающая за сбор данных по выбранному курсу и формированию архива из них, а также формирования файла с информацией о всех доступных для скачивания курсов, который носит название info.json. Ключевыми зависимостями этого компонента, являются:

- archiver - библиотека, для архивирования, в данном случае используются zip-архивы
- sha256 - библиотека для вычисления hash-кода для архива, результат вычисления размещается в поле описания курса в файле info.json и необходим для корректного скачивания курса

Остальные зависимости приведены в файле package.json для этого компонента.

3.6. Файловое хранилище

В качестве реализации файлового хранилища используется Scality S3 Server. Ниже приведен фрагмент кода файла docker-compose.yml для файлового хранилища:

Листинг 3.4. Фрагмент файла docker-compose.yml для компонента Файловое хранилище

```
  scality:
    image: scality/s3server
    ports:
      - 8080:8000
    volumes:
      - ./scality/config.json:/usr/src/app/config.json
```

```
environment:
  SCALITY_ACCESS_KEY_ID: '12340'
  SCALITY_SECRET_ACCESS_KEY: '01234560'
  SSL: 'FALSE'
```

Через среду окружения задаются два ключа доступа. Для компонентов, которые взаимодействуют с файловым хранилищем указываются такие же ключи, после этого для них разрешаются операции на запись.

3.7. Докеризация

При проектировании было выделено 5 докеров под клиентскую часть, серверную, упаковывающее приложение, базу данных и файловое хранилище. Пример docker-файла для упаковщика:

Листинг 3.5. Содержимое Docker-файла для компонента Упаковщик

```
FROM node:8

ENV NODE_ENV development
ENV PORT 8080
ENV MONGO_URL mongodb://localhost:27017/course-editor

WORKDIR /course-packer
ADD . .

RUN npm install

ENTRYPOINT npm start
```

Для запуска контейнеров и организации связей между ними используется docker-compose. В docker-compose прописаны зависимости от двух сторонних docker-образ:

- mongo - образ с базой данных MongoDB
- scality/s3server - образ с файловым хранилищем S3 server, реализующем AWS интерфейс

Запуск приложения осуществляется с помощью команды docker-compose up. После этого скачиваются и устанавливаются все Docker-образы описанные в файле docker-compose.yml и происходит запуск веб-приложения.

3.8. Тестирование

Для тестирования сервера использовались JavaScript-библиотеки Mocha[4] и Chai[5]. Mocha, как основной фреймворк для тестирования и Chai, как библиотека для формирования http-запросов и проверки их результатов. Для вычисления покрытия кода тестами использовалась библиотека nyc.

Листинг 3.6. Пример теста, который проверяет корректность механизма валидации данных

```
it('POST /auth/registration (error: "All fields must be filled in!"
     for 2 fields)', done => {
      chai
      .request(server)
      .post("/auth/registration")
      .type('form')
      .send({
        'login': 'name'
      })
      .end((err, res) => {
        res.should.have.status(200);
        res.should.be.json;
        res.body.ok.should.be.false;
        res.body.error.should.be.equal("All fields must be filled in!
             ");
        res.body.fields.should.have.members(['password', '
             passwordConfirm']);
        done();
      });
});
```

[4] https://mochajs.org/
[5] https://www.chaijs.com/

На сервер отправляется POST запрос по url “/auth/registration”. Запрос содержит форму, форма одно поле. Ожидается, что сервер вернёт 200 код, что будет свидетельствовать об успешной обработке запроса. Тело ответа должно быть в формате JSON и должно содержать сообщение об ошибке, что какое-то поле не заполнно. Также ожидается, что в ответе содержатся идентификаторы полей, которые не устроили сервер.

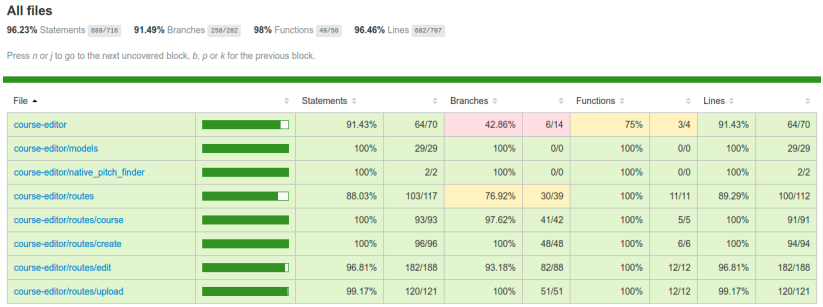
Выделено 104 теста, каждый из которых отвечает за покрытие отведенной функциональности сервера. Тесты объединены в 5 групп:

- Создание пользователя и его аутентификация
- Создание и проверка на доступность для курсов, уроков и задач
- Редактирование и удаление курсов, уроков, задач
- Загрузка медиафайлов(изображений и звуковых файлов)
- Обработка некорректных запросов к серверу

Библиотека для вычисления покрытия кода тестами nyc, позволяет считать покрытие по 4-м метрикам. В качестве результирующего значение берётся средние из посчитаных величин.

В итоге тестирования был получен следующий результат:

Файлы, где покрытие по некоторым метрикам опускается ниже 80%, можно объяснить невозможностью достичь определенных участков кода в связи с тем, что в них используется переключение между тестовым и реальным режимами.



All files

96.23% Statements 689/716 **91.49%** Branches 258/282 **98%** Functions 49/50 **96.46%** Lines 682/707

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
course-editor		91.43%	64/70	42.86%	6/14	75%	3/4	91.43%	64/70
course-editor/models		100%	29/29	100%	0/0	100%	0/0	100%	29/29
course-editor/native_pitch_finder		100%	2/2	100%	0/0	100%	0/0	100%	2/2
course-editor/routes		88.03%	103/117	76.92%	30/39	100%	11/11	89.29%	100/112
course-editor/routes/course		100%	93/93	97.62%	41/42	100%	5/5	100%	91/91
course-editor/routes/create		100%	96/96	100%	48/48	100%	6/6	100%	94/94
course-editor/routes/edit		96.81%	182/188	93.18%	82/88	100%	12/12	96.81%	182/188
course-editor/routes/upload		99.17%	120/121	100%	51/51	100%	12/12	99.17%	120/121

Рис.3.7. Результаты тестирования.

В процессе тестирования были выявлены ошибки связанные с некорректной обработкой запросов. На сервер отправлялись запросы с несуществующим идентификатором курса, из-за неправильной последовательности разбора случался сбой.

Код размещается в gitlab репозитории. После каждого обновления кода в репозитории происходит запуск тестов и вычисляется покрытие кода. Для реализации непрерывной интеграции используется Gitlab CI.

3.9. Результаты

В результате разработки было получено веб-приложение состоящее из 5 докеров (клиент, сервер, база данных, упаковщик, файловое хранилище) и сформирован docker-compose файл, позволяющий разворачивать приложение.

Разработан набор тестов обеспечивающий покрытие кода серверной части свыше 90%. Docker контейнеры и сервис Gitlab CI позволяют запускать тесты и вычислять покрытие, после каждого обновления кода в репозитории.

Проект состоит из 5 компонентов. Серверная часть состоит из

49 файлов, а клиентская из 31 файла. Упаковщик состоит из 13 файлов.

ЗАКЛЮЧЕНИЕ

Работа разделена на 3 главы, отражающие жизненный цикл разработки программного обеспечения: анализ предметной области, заключающийся в обзоре существующих решений, и разработка необходимых требований, предъявляемых к продукту; проектирование архитектуры, включающее в себя выбор технологического стека; реализация конечного продукта с дальнейшим тестированием.

В ходе работы определено место продукта в предметной области. Сформулирована мотивация разработки собственного решения. Цель работы достигнута. Веб-приложение, позволяющее создавать, редактировать и распространять курсы по изучению интонации для пользователей проекта “Study Intonation” разработано успешно. Функциональные и нефункциональные требования, предъявляемые к продукту, выполнены. Освоен процесс разработки одностраничного веб-приложения, получен опыт реализации приложений на технологическом стеке MERN + AWS S3, а также опыт работы с Docker-контейнерами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Alontseva N.V. Kuznetsova Ju.V. Creating a distance program in English, taking into account the specifics of the adult audience. // World of Science. Pedagogy and psychology. — Т. 6, № 2. — Режим доступа: https://mir-nauki.com/PDF/54PDMN218.pdf.
2. Banks Alex, Porcello Eve. Learning React. — O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. — ISBN: 978-1-491-95462-1.
3. Başal Ahmet, Aytan Talat. Using Web 2.0 tools in English language teaching // Conference proceedings. ICT for language learning / libreriauniversitaria. it Edizioni. — 2014. — P. 372.
4. Brown Ethan. Web Development with Node and Express. — O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. — ISBN: 978-1-491-94930-6.
5. Franklin Tom, Harmelen M van. Web 2.0 for content for learning and teaching in higher education. — 2007.
6. Goyal Sumit. E-Learning: future of education // Journal of Education and learning. — 2012. — Vol. 6, no. 4. — P. 239–242.
7. Hahn Evan. Express in Action. — Manning Publications, United States, 2016. — ISBN: 978-1-617-29242-2.
8. Huang Chung-Kai, Chao YC, Lin Chun-Yu. Web 2.0 in and out of the language classroom // Proceedings of the 2008 World CALL: Using Technologies for Language Learning. Foukoka, Japan, q-039. — 2008.
9. Jethro Olojo Oludare, Grace Adewumi Moradeke, Thomas Ajisola Kolawole. E-learning and its effects on teaching and learning in a global age // International Journal of Academic Research in Business and Social Sciences. — 2012. — Vol. 2, no. 1. — P. 203.

10. Pennington Martha C, Rogerson-Revell Pamela. Using Technology for Pronunciation Teaching, Learning, and Assessment // English Pronunciation Teaching and Research. — Springer, 2019. — P. 235–286.
11. Prompt team edited by Mária Hartyányi. Tenegen's pedagogical model competency framework. — 2009.
12. Roles of teachers in e-learning: How to engage students & how to get free e-learning and the future / İlker Yengin, Dilek Karahoca, Adem Karahoca, Ahmet Yücel // Procedia-Social and Behavioral Sciences. — 2010. — Vol. 2, no. 2. — P. 5775–5787.
13. Study Intonation: Mobile Environment for Prosody Teaching / Yurij Lezhenin, Anton Lamtev, Vadim Dyachkov et al. // 2017 3rd IEEE International Conference on Cybernetics (CYBCONF) / IEEE. — 2017. — P. 1–2.
14. Subramanian Vasan. Pro MERN Stack. — apress, 2017. — ISBN: 978-1-4842-2653-7Z.
15. Task Module Four Assessment. Discourse Intonation: To Teach or not to Teach?
16. Thiruvengadam Pushpanathan. The Role of a Teacher in facilitating E-Learning. — 2012. — Access mode: `https://www.researchgate.net/publication/224946774`.
17. Valls Lucía S. English Intonation and its Prominent Role in Teaching Communication. — Access mode: `https://www.academia.edu/19886009/English_Intonation_and_its_Prominent_Role_in_Teaching_Communication`.
18. Web 2.0. — Access mode: `https://www.techopedia.com/definition/4922/web-20`.
19. Бородицкая ГП, Пазюк КТ. Актуальность дистанционного образования в России // Ученые заметки ТОГУ. — 2017. — Т. 8, № 1. — С. 387–389.

20. Ваганова Ольга Игоревна, Гладкова Марина Николаевна, Трутанова Александра Валерьевна. Электронное обучение как средство организации самостоятельной работы студентов // Балтийский гуманитарный журнал. — 2017. — Т. 6, № 2 (19).
21. Изучаем Node.js. — СПб.: Питер, 2014. — ISBN: 978-5-496-00356-8.
22. Проект ШЛЮПА. — Режим доступа: http://ru.knowledgr.com/03194570/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%A8%D0%9B%D0%AE%D0%9F%D0%90.
23. С.В. Праводелов. Преимущества дистанционного обучения и его виды // Современное образование. — 2015. — № 2. — С. 70–79. — Режим доступа: https://nbpublish.com/library_read_article.php?id=14207.
24. Ярашева Д, Ибрагимова С. Информационные технологии в обучении иностранному языку // Современное образование (Узбекистан). — 2014. — № 5.

ПРИЛОЖЕНИЕ 1. ЛИСТИНГИ

Листинг 7. Компонент - Сервер, файл course/course.js, содержащий обработку 2 запросов

```
const express = require("express"),
      router = express.Router(),
      request = require('request'),
      models = require("../../models");

router.get("/:courseId", async (req, res, next) => {
  const userLogin = req.session.userLogin;
  const userId = req.session.userId;

  const courseId = req.params.courseId;

  if (!courseId.match(/^[0-9a-fA-F]{24}$/)) {
    const err = new Error("Not Found");
    err.status = 404;
    next(err);
  } else {
    const course = await models.Course.findById(courseId);
    if (!course) {
      const error = new Error("Not Found");
      error.status = 404;
      next(error);
    } else if (
      !course.published &&
      (!userLogin || !userId || userId != course.owner)
    ) {
      res.json({
        ok: false,
        error: 'Forbidden',
      });
    } else {
      res.json({
        ok: true,
        course,
      });
    }
  }
});
```

```
router.get("/publish/:courseId", async (req, res, next) => {
  const userLogin = req.session.userLogin;
  const userId = req.session.userId;

  const courseId = req.params.courseId;

  if (!courseId.match(/^[0-9a-fA-F]{24}$/)) {
    const err = new Error('Not Found');
    err.status = 404;
    next(err);
  } else {
    const course = await models.Course.findById(courseId);

    if (!course) {
      const error = new Error('Not Found');
      error.status = 404;
      next(error);
    } else if (
      !course.published &&
      (!userLogin || !userId || userId != course.owner)
    ) {
      res.json({
        ok: false,
        error: 'Forbidden',
      });
    } else {
      request.post('http://course-packer:8000/course').form({id:
          courseId});
      res.json({
        ok: true,
        url: `/uploads/${courseId}.zip`,
      })
    }
  }
});

module.exports = router;
```

Листинг 8. Компонент - Сервер, файл edit/course.js, содержащий обработку 2 запросов

```
const express = require("express"),
      router = express.Router(),
      models = require("../../models"),
```

```
      util = require("../utils");

router.post('/', async (req, res, next) => {
  const userLogin = req.session.userLogin;
  const userId = req.session.userId;

  const courseId = req.body.id;
  const course = await models.Course.findById(courseId);
  if (!course) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
  } else if (!userLogin || !userId || userId != course.owner) {
    res.json({
      ok: false,
      error: 'Forbidden',
    });
  } else {
    let {
      title,
      description,
      complexity,
      category,
      authors,
      published
    } = req.body;

    if (!title) {
      title = course.title;
    }
    if (!description) {
      description = course.description;
    }
    if (!complexity) {
      complexity = course.complexity;
    }
    if (!category) {
      category = course.category;
    }
    if (!authors) {
      authors = course.authors;
    }
    if (!published) {
      published = course.published;
    }
```

```
    const newCourse = await models.Course.findOneAndUpdate(
      {
        _id: course.id,
        owner: userId
      },
      {
        title,
        description,
        complexity,
        category,
        authors,
        owner: userId,
        published
      },
      { new: true }
    );

    if (!newCourse) {
      res.json({
        ok: false,
        error: 'Something went wrong'
      });
    } else {
      res.json({
        ok: true
      });
    }
  }
});

router.delete('/', async (req, res, next) => {
  const userLogin = req.session.userLogin;
  const userId = req.session.userId;

  const courseId = req.body.id;
  const course = await models.Course.findById(courseId);
  if (!course) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
  } else if (!userLogin || !userId || userId != course.owner) {
    res.redirect("/");
  } else {
    var removedTasksOK = true;
```

```
    const lessons = await models.Lesson.find({ course: courseId });
    lessons.forEach(async lesson => {
      const tasks = await models.Task.find({ lesson: lesson.id });
      tasks.forEach( task => {
        util.removeUploadsFromS3(task.sound);
        util.removeUploadsFromS3(task.pitch);
      });
      const removedTasks = await models.Task.deleteMany({ lesson:
          lesson.id });
      removedTasksOK = removedTasksOK && removedTasks.ok;
    });
    const removedLessons = await models.Lesson.deleteMany({ course:
        course.id });
    util.removeUploadsFromS3(course.logo);
    const removedCourse = await models.Course.findByIdAndRemove(
        course.id);

    if (!removedCourse || !removedLessons.ok || !removedTasksOK) {
      res.json({
        ok: false,
        error: 'Something went wrong'
      });
    } else {
      res.json({
        ok: true
      });
    }
  }
});

module.exports = router;
```

Листинг 9. Компонент - Сервер, файл image.js содержащий обработку загрузки изображений

```
const express = require("express"),
      router = express.Router(),
      path = require("path"),
      multer = require("multer"),
      multerS3 = require('multer-s3'),
      AWS = require("aws-sdk"),
      config = require("../../config"),
      models = require("../../models"),
      util = require("../utils");
```

```
const s3 = new AWS.S3(config.AWS_PARAM);
const storage = multerS3({
  s3: s3,
  bucket: config.DESTINATION,
  acl: 'public-read',
  contentType: multerS3.AUTO_CONTENT_TYPE,
  key: async (req, file, cb) => {
    const userLogin = req.session.userLogin;
    const userId = req.session.userId;

    const courseId = req.body.courseId;
    const course = await models.Course.findById(courseId);

    if (!course) {
      let err = new Error('Not Found');
      err.code = 'NOCOURSE';
      cb(err);
    } else if (!userLogin || !userId || userId !=
      course.owner) {
      let err = new Error('Is not allowed');
      err.code = 'NOTALLOWED';
      cb(err);
    } else {
      if (course.logo !== 'none') {
        util.removeUploadsFromS3(course.logo);
      }
      const filePath = Date.now() + path.extname(file.originalname);
      await models.Course.findOneAndUpdate(
        {
          _id: course.id,
          owner: userId
        },
        {
          logo: filePath
        },
        { new: true }
      );
      cb(null, filePath);
    }
  }
});

const upload = multer({
  storage,
```

```
  fileFilter: (req, file, cd) => {
    const ext = path.extname(file.originalname);
    if (ext !== '.jpg' && ext !== '.jpeg' && ext !== '.png') {
      const err = new Error('Extention');
      err.code = 'EXTENTION';
      return cd(err);
    }
    cd(null, true);
  },
  limits: {
    fileSize: 3 * 1024 * 1024,
  },
}).single('file');

router.post('/', (req, res) => {
  upload(req, res, err => {
    let error = '';
    if (err) {
      console.log(err);
      if (err.code === 'EXTENTION') {
        error = 'Only jpeg and png!';
      }
      if (err.code === 'NOCOURSE') {
        error = 'Nothing found!';
      }
      if (err.code === 'NOTALLOWED') {
        error = 'Invalid auth!';
      }
      if (err.code === 'LIMIT_FILE_SIZE') {
        error = 'Limit file size!';
      }
      if (err.code === 'NoSuchBucket') {
        util.createBucketS3(config.DESTINATION);
        error = 'No such Bucket, pls, try again.';
      }
    }
    res.json({
      ok: !err,
      error
    });
  });
});

module.exports = router;
```

Листинг 10. Компонент - Сервер, файл image.js содержащий обработку загрузки изображений

```
FROM node:8

ENV NODE_ENV development
ENV PORT 8080
ENV MONGO_URL mongodb://localhost:27017/course-editor

WORKDIR /course-packer
ADD . .

RUN npm install

ENTRYPOINT npm start
```

Листинг 11. Компонент - Сервер, файл my.js содержащий обработку запроса на курсы, принадлежащие пользователю

```
const express = require("express"),
      router = express.Router(),
      models = require("../models");

router.get("/", async (req, res) => {
  try {
    const userId = req.session.userId;
    const userLogin = req.session.userLogin;

    const courses = await models.Course.find({ owner: userId }).sort
        ({
      createdAt: -1
    });

    if (!userLogin || !userId) {
      res.json({
        ok: false,
        error: 'Forbidden',
      })
    } else {
      res.json({
        ok: true,
        courses,
      });
```

```
    }
  } catch (error) {
    throw new Error("Server Error");
  }
});

module.exports = router;
```

Листинг 12. Компонент - Сервер, файл config.js содержащий конфигурационные параметры для серверной части

```
const dotenv = require("dotenv");
const path = require("path");

const root = path.join.bind(this, __dirname);
dotenv.config({ path: root(".env") });

let mongoURL = "";
if (process.env.NODE_ENV == "test") {
  mongoURL = process.env.TEST_MONGO_URL;
} else {
  mongoURL = process.env.MONGO_URL;
}

const AWS_PARAM = {
  accessKeyId: process.env.ACCESS_KEY_ID,
  secretAccessKey: process.env.SECRET_ACCESS_KEY,
  endpoint: 'scality:8000',
  sslEnabled: false,
  s3ForcePathStyle: true
};

module.exports = {
  MONGO_URL: mongoURL,
  SESSION_SECRET: process.env.SESSION_SECRET,
  DEBUG: process.env.NODE_ENV === "development",
  PORT: process.env.PORT || 3000,
  DESTINATION: "uploads",
  AWS_PARAM
};
```

Листинг 13. Компонент - Сервер, ключевой файл app.js, обеспевивающий подключение основных модулей и перенаправляющий обработку запросов

```
const express = require("express"),
      path = require("path"),
      favicon = require("static-favicon"),
      logger = require("morgan"),
      cookieParser = require("cookie-parser"),
      bodyParser = require("body-parser"),
      proxy = require("express-http-proxy");

const index = require("./routes/index"),
      create = require("./routes/create"),
      edit = require("./routes/edit"),
      auth = require("./routes/auth"),
      course = require("./routes/course/course"),
      lesson = require("./routes/course/lesson"),
      task = require("./routes/course/task"),
      upload = require("./routes/upload"),
      myCourses = require("./routes/my"),
      info = require("./routes/info");

const staticAsset = require("static-asset");

const config = require("./config");

const cookieSession = require('cookie-session');

// database
const mongoose = require("mongoose");
mongoose.set("debug", config.DEBUG);
mongoose.connect(
  config.MONGO_URL,
  { useNewUrlParser: true }
);

const db = mongoose.connection;
db.on("error", console.error.bind(console, "Connection error:"));
db.once("open", function() {
  if (config.DEBUG) {
    console.log("Connection to database open.");
  }
});
```

```
// express
const app = express();

app.use(cookieSession({
  name: 'session',
  keys: [config.SESSION_SECRET]
}));

// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");

if (process.env.NODE_ENV !== 'test') {
  app.use(logger("dev"));
}

app.use(favicon());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }))
app.use(cookieParser());
app.use(staticAsset(path.join(__dirname, "public")));
app.use(express.static(path.join(__dirname, "public")));

app.use('/uploads', proxy('scality:8000', {
  proxyReqPathResolver: (req) => {
    return '/uploads' + req.url;
  }
}));

app.use("/", index);
app.use("/create", create);
app.use("/edit", edit);
app.use("/auth", auth);
app.use("/course", course);
app.use("/lesson", lesson);
app.use("/task", task);
app.use("/upload", upload);
app.use("/my", myCourses);
app.use("/info", info);

/// catch 404 and forwarding to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
```

```
  next(err);
});

/// error handlers

// development error handleenvr
// will print stacktrace
if (app.get('env') === 'development') {
  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.json({
      message: err.message,
      error: err
    });
  });
}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.json({
    message: err.message,
    error: {}
  });
});

module.exports = app;
```

Листинг 14. Компонент - Клиент, файл App.js

```
import React, {Component} from 'react';
import CourseEditor from './components/CourseEditor';
import { BrowserRouter } from 'react-router-dom';

class App extends Component {

  render() {
    return (
      <div className="App">

        <BrowserRouter>

          <CourseEditor />
```

```
        </BrowserRouter>

      </div>
    );
  }
}

export default App;
```

Листинг 15. Компонент - Клиент, файл Courses.js

```
import React, {Component} from 'react';
import Grid from '@material-ui/core/Grid';
import Typography from '@material-ui/core/Typography';
import { withStyles } from '@material-ui/core/styles';
import Fab from '@material-ui/core/Fab';
import AddIcon from '@material-ui/icons/Add';
import { Link } from 'react-router-dom';
import Card from './MediaCard';
import utils from '../utils';

const styles = theme => ({
  title: {
    marginTop: theme.spacing(1),
    marginBottom: theme.spacing(4),
    marginLeft: theme.spacing(3),
  },
  fab: {
    margin: 0,
    top: 'auto',
    right: 20,
    bottom: 20,
    left: 'auto',
    position: 'fixed',
    'z-index': 999,
  },
});

class Courses extends Component {
  state = {courses: []};

  componentDidMount() {
    utils.callApi('/index')
```

```
    .then(res => this.setState({courses: res.courses}))
    .catch(err => console.log(err));
}

CoursesCards = () => {
  if (this.state.courses.length === 0) {
    return (
      <div>
        <p>
          There are no courses, but you should rejoice!
        </p>
      </div>
    )
  } else {
    return (
      <Grid container spacing={4}>
        {this.state.courses.map( course =>
          <Card key={course.id} course={course}/>
        )}
      </Grid>
    )
  }
}

render() {
  return (
    <div className="container">
      <Typography component="h1" variant="h3" color="inherit"
          className={this.props.classes.title}>
        Courses
      </Typography>

      {
        this.props.auth ?
          <Link
            to='/create/course'
            className={this.props.classes.fab}
          >
            <Fab
              size="medium"
              color="secondary"
              aria-label="Add"
            >
                <AddIcon />
            </Fab>
```

```
            </Link>
          :
            null
        }

        <this.CoursesCards className={this.props.classes.lol}/>

      </div>
    );
  }
}

export default withStyles(styles)(Courses);
```