Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

КУРСОВАЯ РАБОТА

по дисциплине «Распределённые системы обработки информации. Часть 2»

**Веб-сайт «Образовательный портал для обмена сайтами и их решениями»**

|  |  |
| --- | --- |
| Студентка | А.Ю. Лазовик |
| Руководитель | А. Н. Атрощенко |

Минск

2018

**Содержание:**

Введение

Описание бизнес-логики приложения

Используемые технологии

Архитектура приложения

Детали реализации приложения

Заключение

Используемая литература

**Введение**

Задачей крсового проекта является проектирование и реализация веб приложения. Для реализации бизнес-логики или серверной части приложения будет использоваться JavaScript. Для реализации клиентской части приложения будут использоваться такие технологии как JavaScript, HTML, CSS. Приложение будет выполнено как функцтонирующее веб-приложение с самостоятельно установленным веб-сервером в архитектуре Model-View-Controller с использованием технологии Ajax и AMD-концепций.

Веб-сайт будет представлять из себя портал для для обмена задачами и их решениями. У каждого пользователя будет своя учётная запись, для получения которой необходимо зарегистрироваться. После авторизации у пользователя будет возможность редактировать свою учётную запись и обмениваться задачами с другими пользователями. Каждый пользователь сможет разместить свою задачу, а также решить задачи других пользователей. На основании решённых задач будет строится рейтинг пользователей.

**Описание бизнес-логики приложения**

Задача этого раздела описать на высоком уровне бизнес-задачи, которые будет решать наше приложение.

С точки зрения пользователя приложение будет обладать следущей функциональностью:

1. Регистрация пользователя
2. Авторизация пользователя
3. Просмотр и редактирование профиля пользователя
4. Публикация задачи пользователя
5. Решение задачи другого пользователя
6. Просмотр рейтинга пользователей

Рассмотрим каждый пункт подробнее. При этом создадим mockups для будущего приложения. Mockups это наброски клиентской части приложения. Это не готовое представление клиентской части, мы будем использовать их как ориентир в процессе реализации.

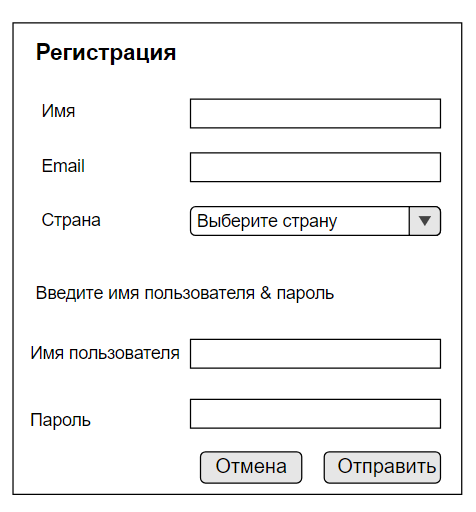
**Регистрация пользователя**

Во время регистрации пользователю показывается форма, которую он должен заполнить, для дальнейшей авторизации в системе, а также для того, чтобы его могли отличить другие пользователи. Для регистрации пользователю неоходимо заполнить следующие поля:

* Имя
* Email
* Страна(выбор из списка)
* Имя пользователя(уникальный идентификатор пользователя)
* Пароль

После заполнения формы и клика на кнопку **Отправить** производится валидация параметров формы: **email** поле должно иметь формат email адреса, например example@gmail.com, поле **Пароль** должно содержать более 6 символов, а также все поля формы должны быть не пустыми. После отправки запроса на сервер, данные пользователя сохраняются в базе дынных.

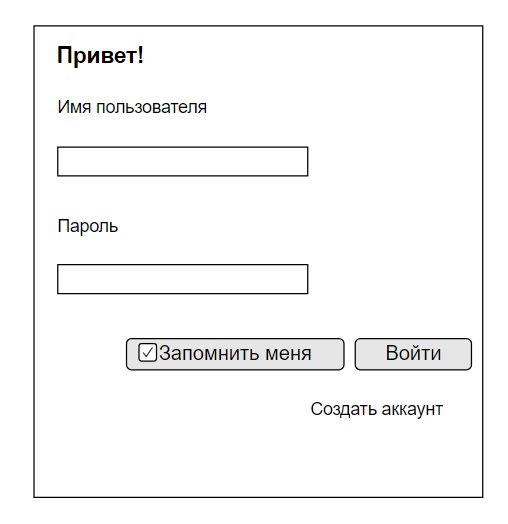
Таким образом страница регистрации будет выглядеть следующим образом:



**Авторизация пользователя**

Для того, чтобы пользватель мог пользоваться фунционалом приложения нам необходимо аторизовать его. Для этого пользователю необходимо ввести имя пользователя и пароль, которые были использованы при регистрации. После отправки данных на сервер мы делаем запрос в базу данных, чтобы убедиться, что пользователь существует. На форме присутствует чекбокс **Запомнить меня**, если пользователь при авторизации нажмёт этот чекбокс, то последующие его входы в систему будут происходить автоматически.

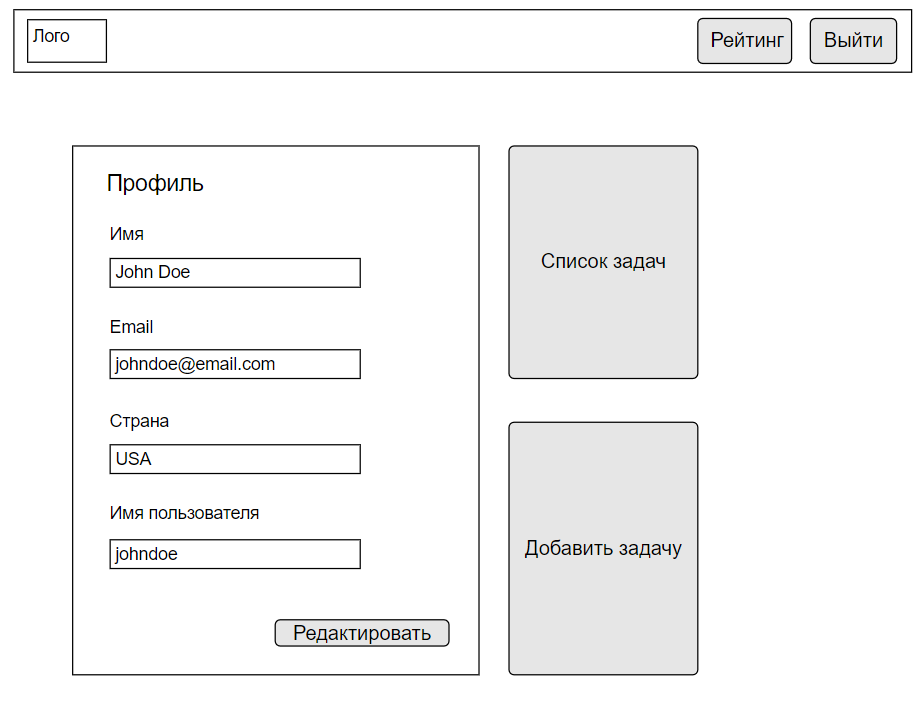
Страница авторизации выглядит следующим образом:



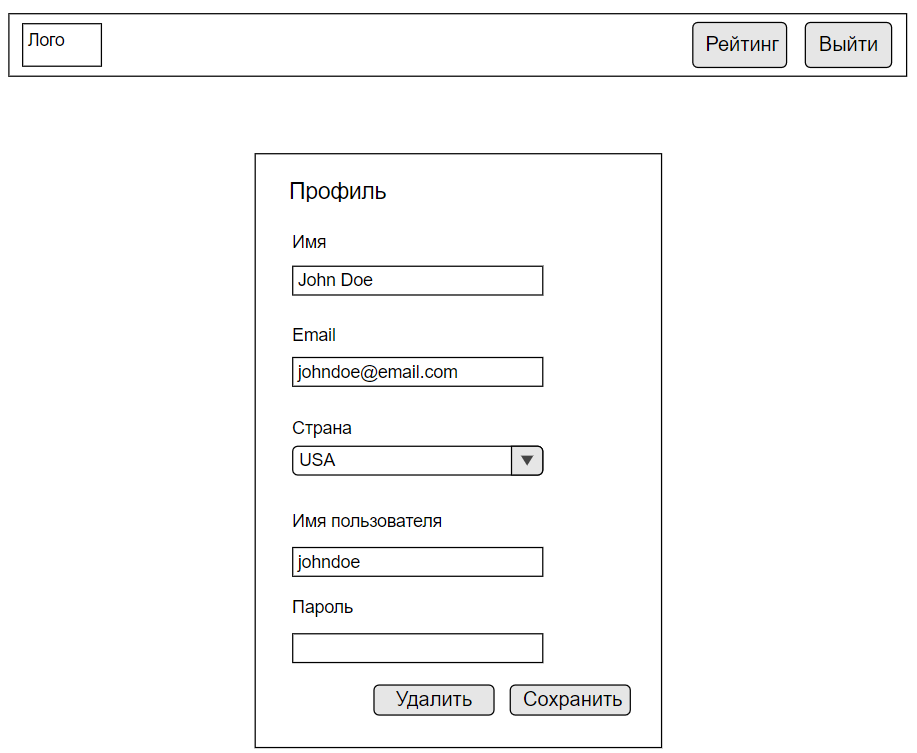
**Просмотр и редактирование профиля пользователя**

После успешной авторизации пользователя, приложение пернапрявляет пользователя на страницу с информацией о его профиле. Также на странице у пользователя будет возможность добавить свою задачу, посмотреть список всех задач, а также рейтинг всех пользователей.

Страница будет выглядеть следующим образом:

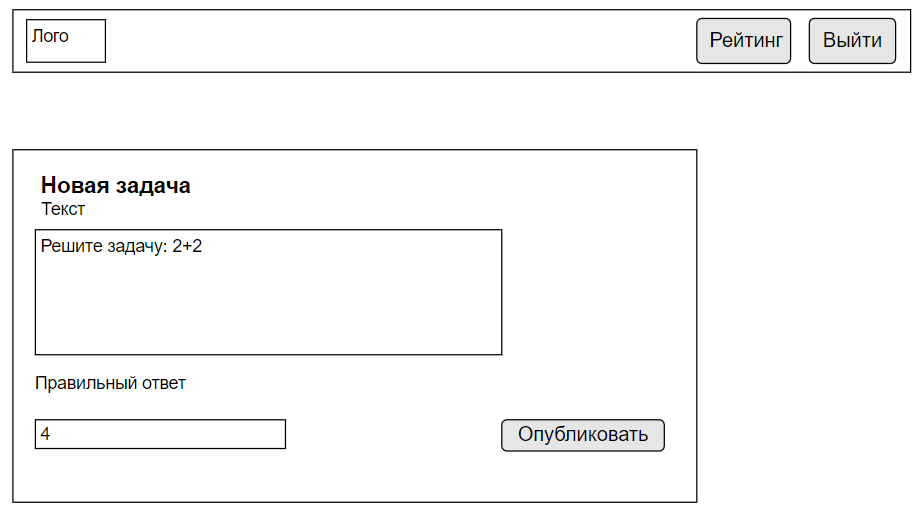


Страница для редактирования профиля будет выглядеть следующим образом:



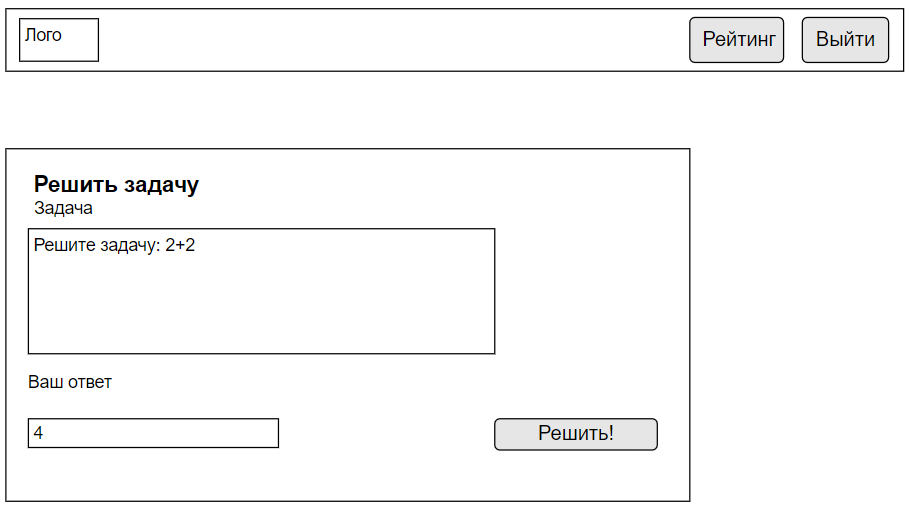
**Публикация задачи пользователя**

После нажатия на кнопку **Добавить задачу**, для пользователя открывается новре страница, на которой он может опублтковать задачу, которую могут решить другие пользователи. Пользователь должен указать текст, а также правильный ответ.



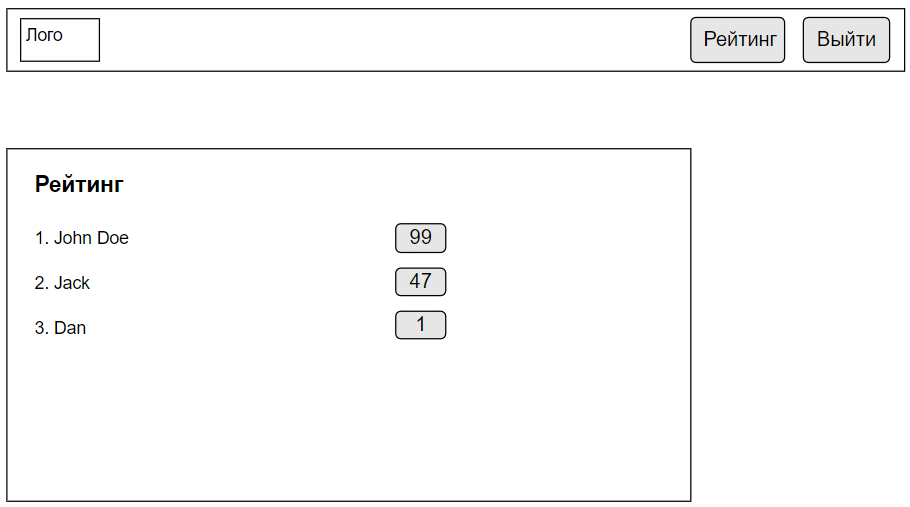
**Решение задачи другого пользователя**

Данная страница будет похожа на страницу публикации задачи. После того как пользователь нажмёт на кнопку **Решить!** будет всплывать модальное окно с результатом решения – правильно или не правильно.



**Просмотр рейтинга пользователей**

Данная страница будет отображать список всех пользователей с количеством решённых задач в порядке убывания, т.е пользователи с наибольшим количеством решённых задач будут вверху списка.



**Используемые технологии**

Как уже упоминалось раннее для клиентской и серверной части приложения будет использоваться JavaScript.

JavaScript -  мультипарадигменный язык программирования. Поддерживает объектноориентированный, императивный и функциональный стили. Является реализацией языка ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений.

Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

В качастве сервера приложения мы будем использовать **Node.js**.

**Node** или **Node.js** — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Ссылка на сайт **https://nodejs.org/en/**

В качестве веб фреймворка будем использовать **Express.js**

**Express.js**, или просто Express, каркас web-приложений для Node.js, реализованный как свободное и открытое программное обеспечение под лицензией MIT. Он спроектирован для создания веб-приложений и API. Де-факто является стандартным каркасом для Node.js.

Ссылка на сайт **http://expressjs.com/**

Для простоты написания и поддрежки html будем использовать Jade. Ссылка на сайт **http://jade-lang.com/**

Для слоя доступа к данным будем испоьзовать **MongoDB.**

**MongoDB** (от англ. *humongous* — *огромный*) — документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Ссылка на сайт **https://www.mongodb.com/**

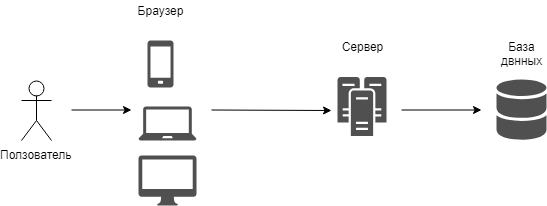
**jQuery** — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Сейчас разработка jQuery ведётся командой jQuery во главе с Джоном Резигом.

**Архитектура приложения**

Веб-сайт представляет из себя **клиент-серверное** приложение.

**Клиент-сервер** - вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

В самом простом случае архитектура клиент-серверного приложения выглядит следующим образом.



Пользователь взаимодействует с клиентским приложением(в данном случае **браузером**), которое посылает запросы на сервер, который в свою очередь использует базу данных для хранения необходимой информации.

Будем использовать самый популярный шаблон для разработки – **Model-View-Controller**. Данный шаблон позволяет рзделить приложение на три отдельные несвязаные части – модель, представление и контроллер. Model-View-Controller позволяет декомпозировать основные компоненты ,что позволяет эффективно переиспользовать код и разрабатывать эти компоннеты параллельно нескольким разработчиками.

Диаграмма компонентов выглядит следующим образом.



Взаимодействие клиента и сервера будет происходит по **HTTP** протоколу.

**HTTP** (англ. *HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

* Потребителей (клиентов), которые инициируют соединение и посылают запрос;
* Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

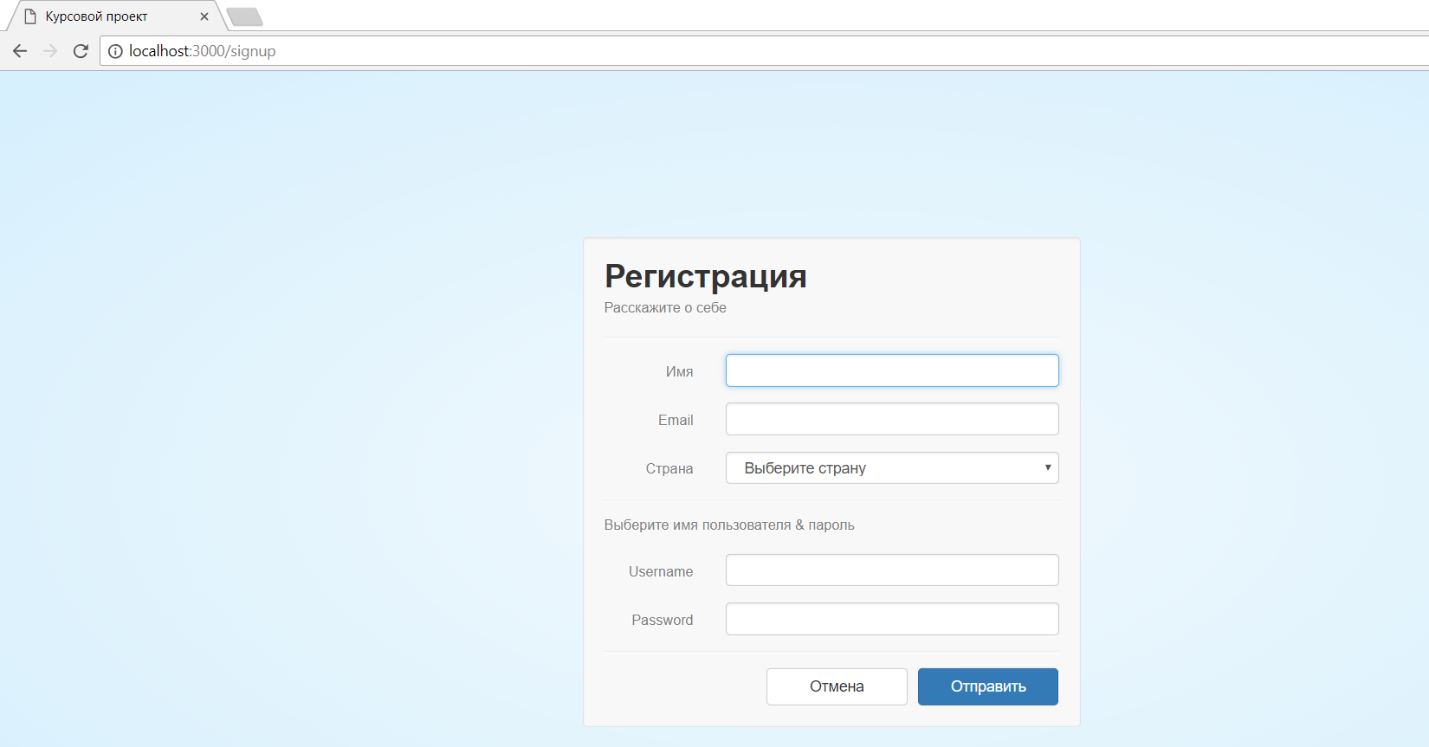
HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов. В 2006 году — в Северной Америке доля HTTP-трафика превысила долю P2P-сетей и составила 46 %, из которых почти половина — это передача потокового видео и звука.

HTTP используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP, XML-RPC, WebDAV.

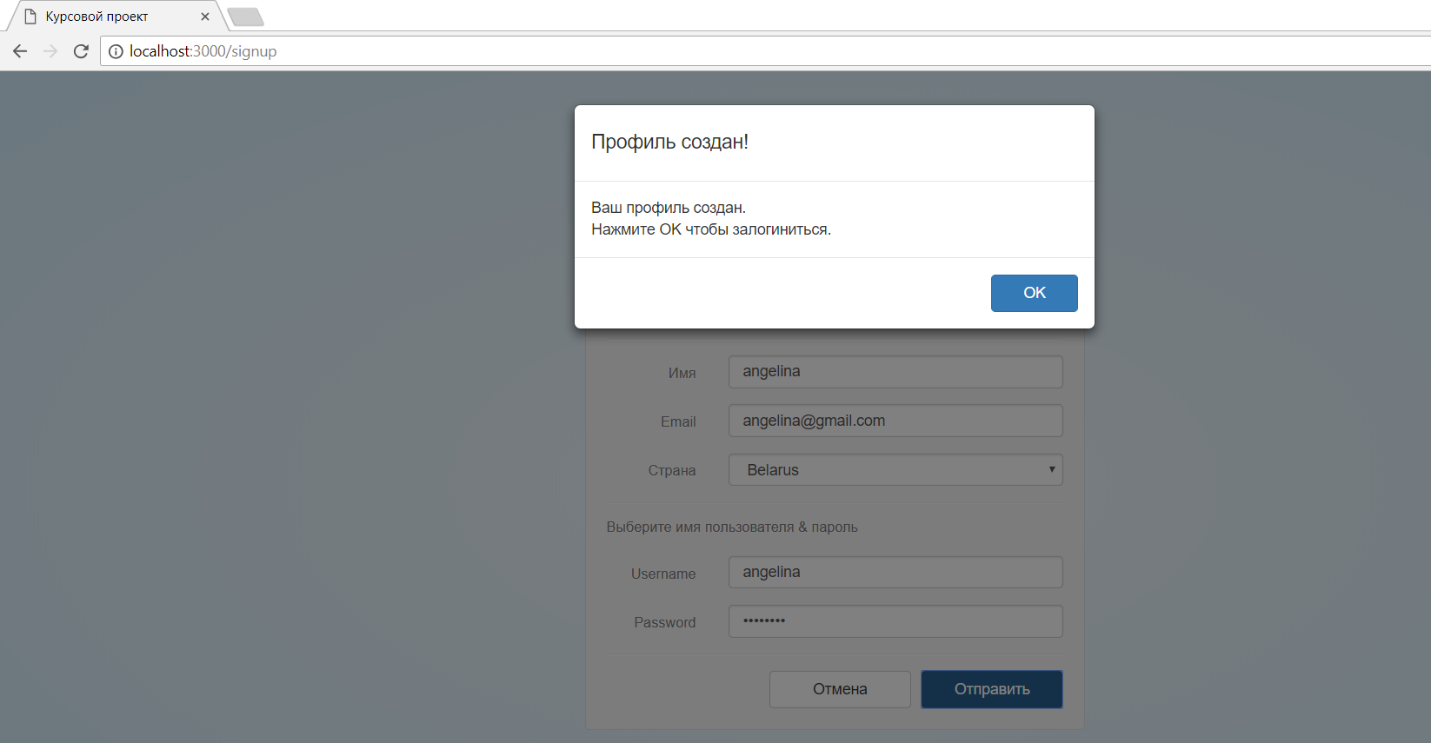
**Детали реализации приложения.**

В этой главе коротко опишем детали реализации приложения. Будут описаны клиентская и серверная части приложния, представлены скриншоты работающего приложения. В конце главы опишем как скачать исходный код и запустить приложение.

Начнём со страницы регистрации пользователя.



Зарегистрируем нового пользователя с именем angelina. После заполнения формы и отправки на сервер для пользователя всплывает модальное окно об успешной регистрации. После нажатия на кнопку ОК пользователь переадресовывается на страницу авторизации.



Все http запросы приходящие от клиента обрабаытваются в файле **routes.js**. В данном случае для обработки post запросы формы регистрации пользователя отвечает следующий код:

**app.post('/signup', function(req, res){**

**AM.addNewAccount({**

**name : req.body['name'],**

**email : req.body['email'],**

**user : req.body['user'],**

**pass : req.body['pass'],**

**country : req.body['country']**

**}, function(e){**

**if (e){**

**res.status(400).send(e);**

**} else{**

**res.status(200).send('ok');**

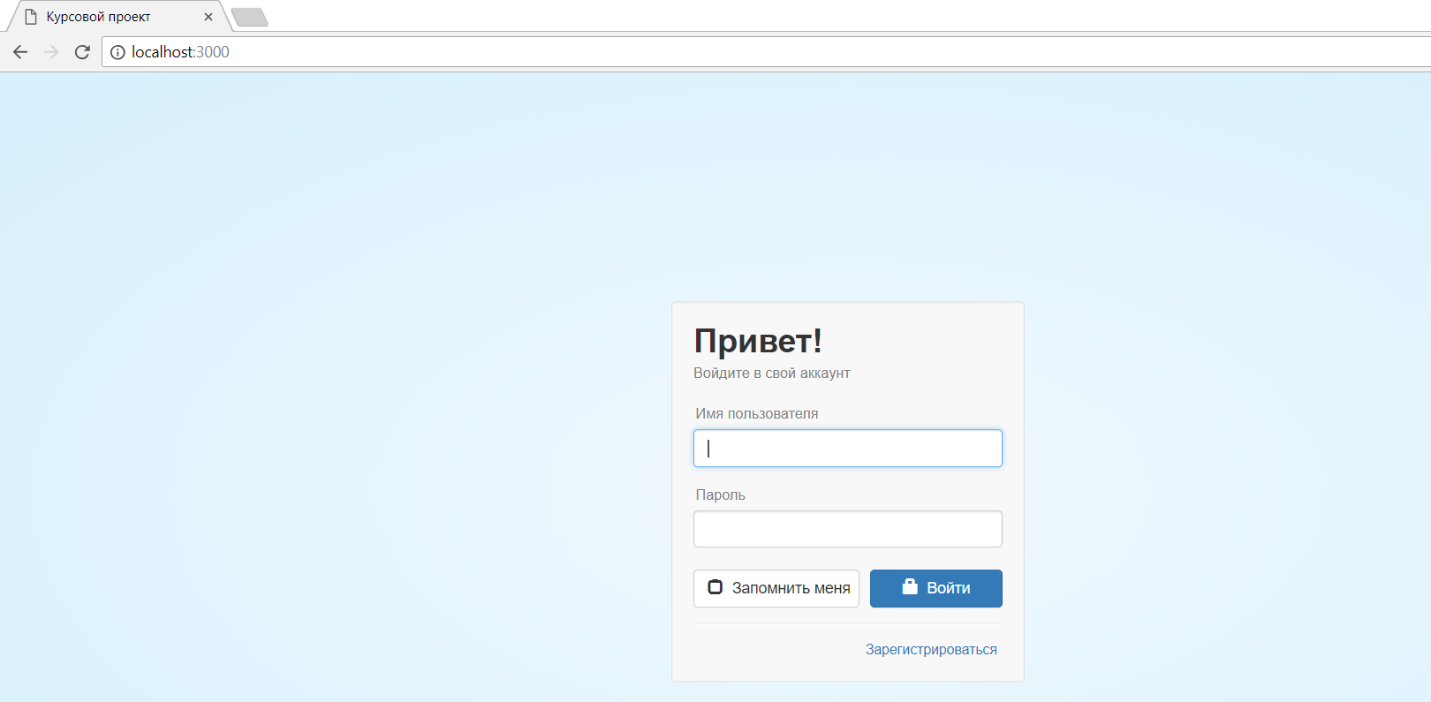
**}**

**});**

**});**

Здесь мы достаём нужные нам атрибуты из тела запроса и сохраняем в **accounts** коллекцию базы данных, используя метод addNewUser.

Далее перейдём к странице авторизации пользователя. Она выглядит следующим образом:



Вводим имя пользователя и пароль, которые были указаны при регистрации.

Post запрос формы авторизации обрабатывает следующи код:

**app.post('/', function(req, res){**

**AM.manualLogin(req.body['user'], req.body['pass'], function(e, o){**

**if (!o){**

**res.status(400).send(e);**

**} else{**

**req.session.user = o;**

**if (req.body['remember-me'] == 'true'){**

**res.cookie('user', o.user, { maxAge: 900000 });**

**res.cookie('pass', o.pass, { maxAge: 900000 });**

**}**

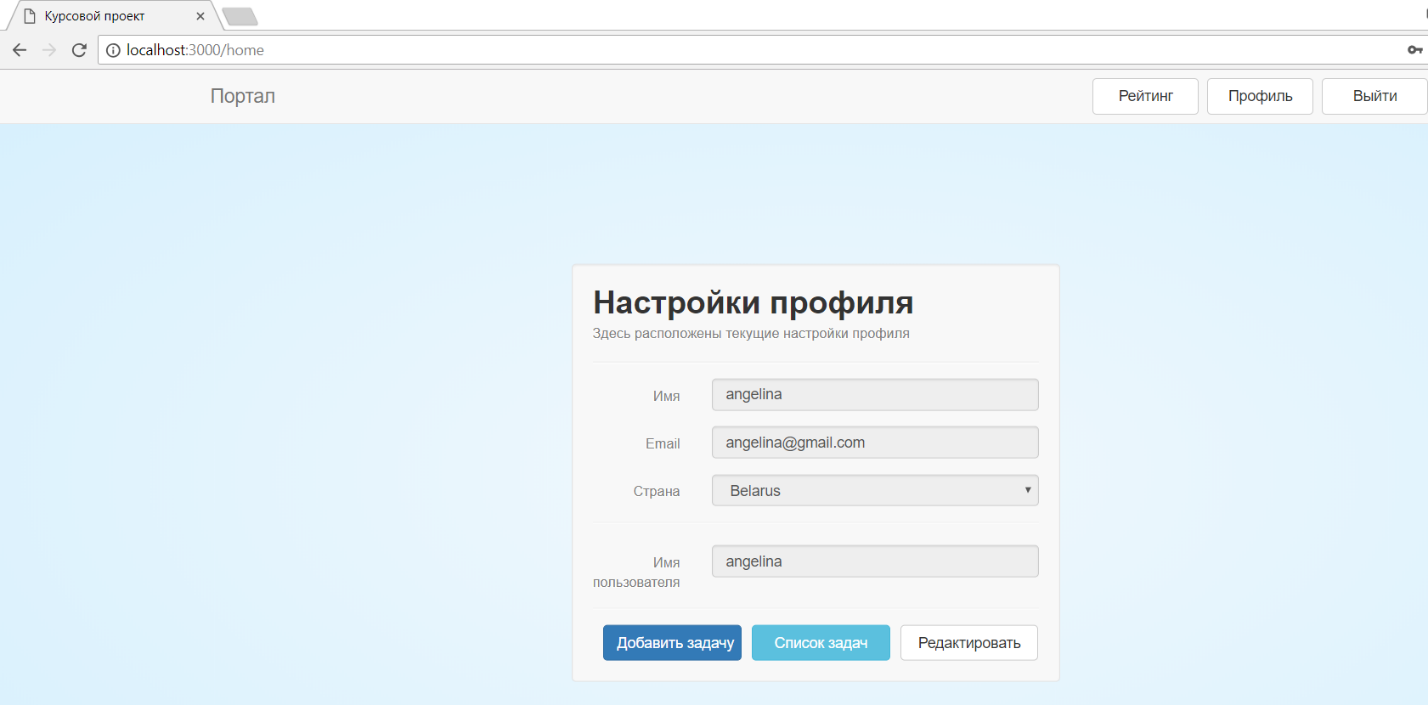
**res.status(200).send(o);**

**}**

**});**

**});**

В случае успешного входа, пользователь переадресовывается на страницу с информацией о его профиле.



Код, который отображает страницу с информацией о пользователе

**app.get('/home', function(req, res) {**

**if (req.session.user == null){**

**// if user is not logged-in redirect back to login page //**

**res.redirect('/');**

**} else{**

**res.render('home', {**

**title : 'Профиль',**

**countries : CT,**

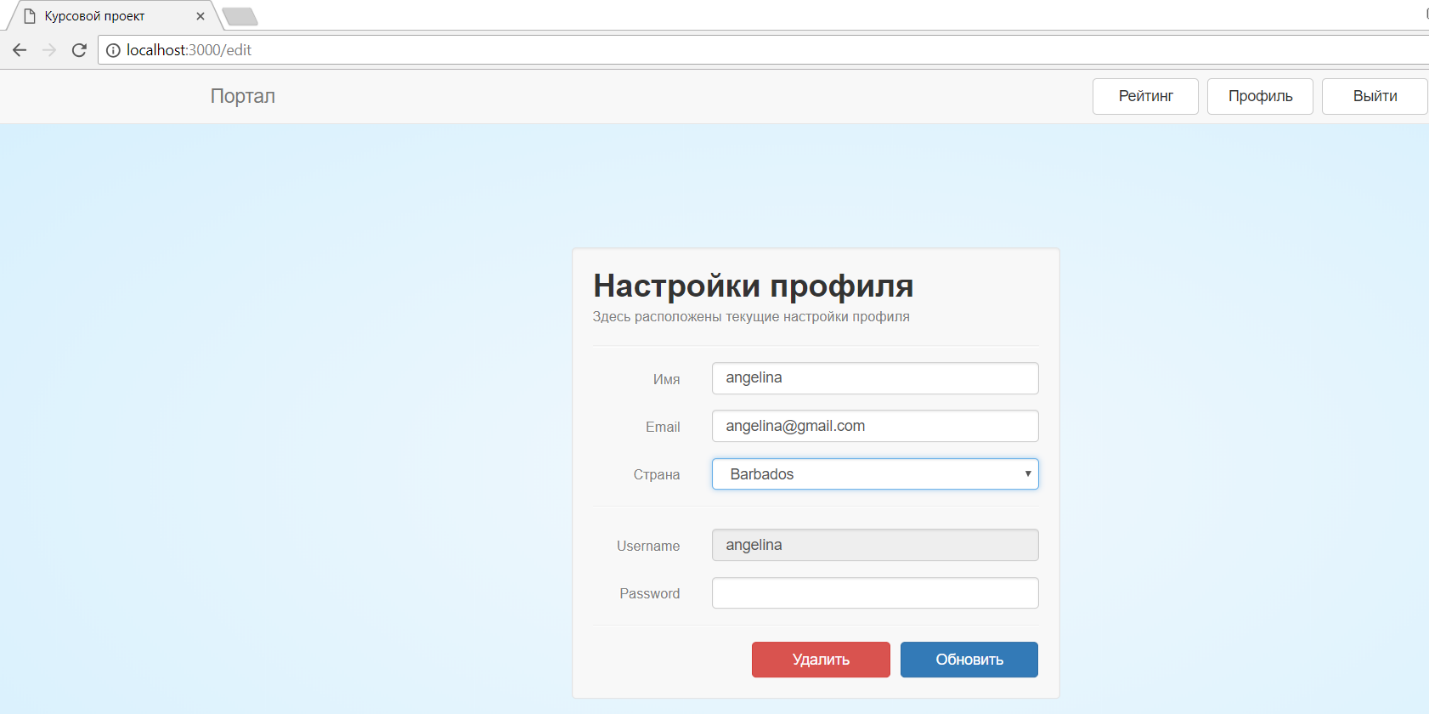
**udata : req.session.user**

**});**

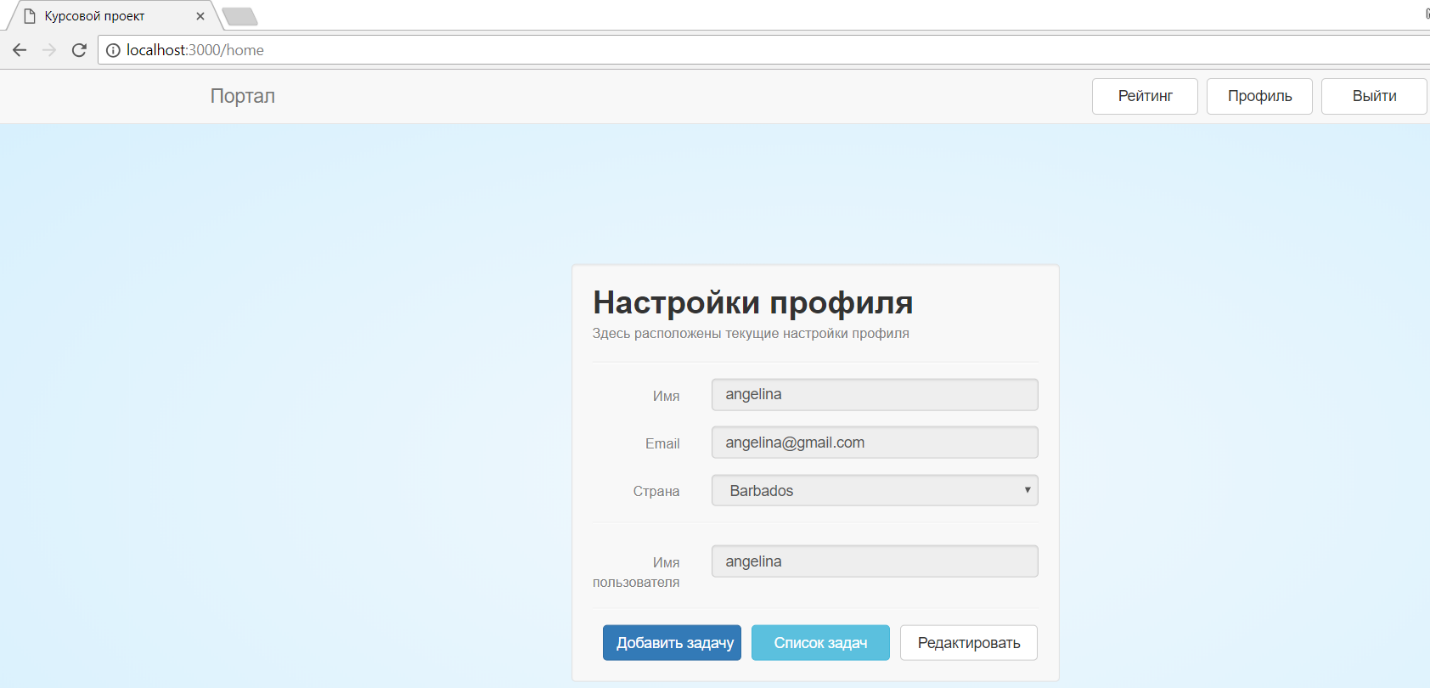
**}**

**});**

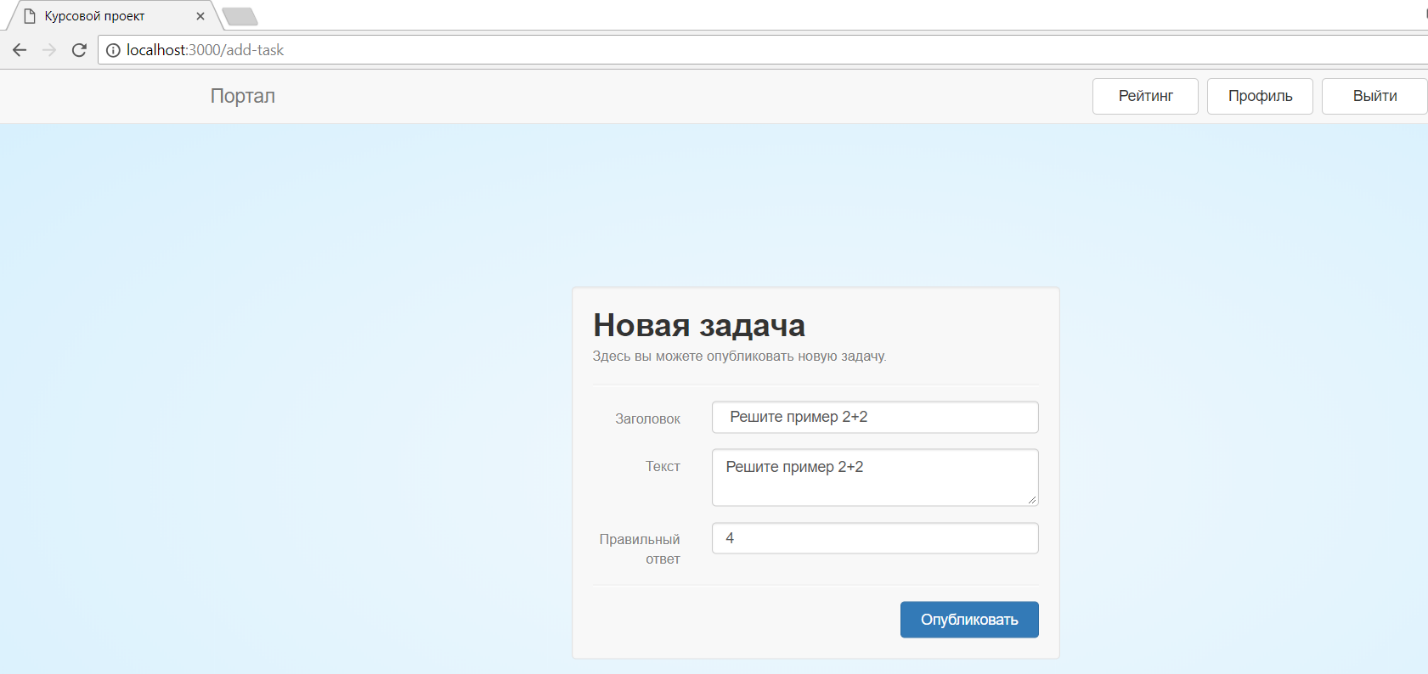
Чтобы изменить профиль пользователю необходимо нажать кнопку **Редактировать**. Изименить например страну пользователя



После обновления страница с информацие пользователя будет выглядеть следующим образом:



Как видно, страна изменилась. Также на этой странице у пользователя есть фозможность добавить задачу, нажав на кнопку **Добавить задачу**. Добавим новую задачу.



После нажатия на кнопку данные формы отправляются на сервер, где обрабатываются следующим кодом

**app.post('/add-task', function(req, res) {**

**taskManager.addNewTask({**

**title : req.body['title'],**

**text : req.body['text'],**

**correct\_answer : req.body['correct-answer'],**

**user : req.session.user**

**}, function(e){**

**if (e){**

**res.status(400).send(e);**

**} else{**

**res.status(200).send('ok');**

**}**

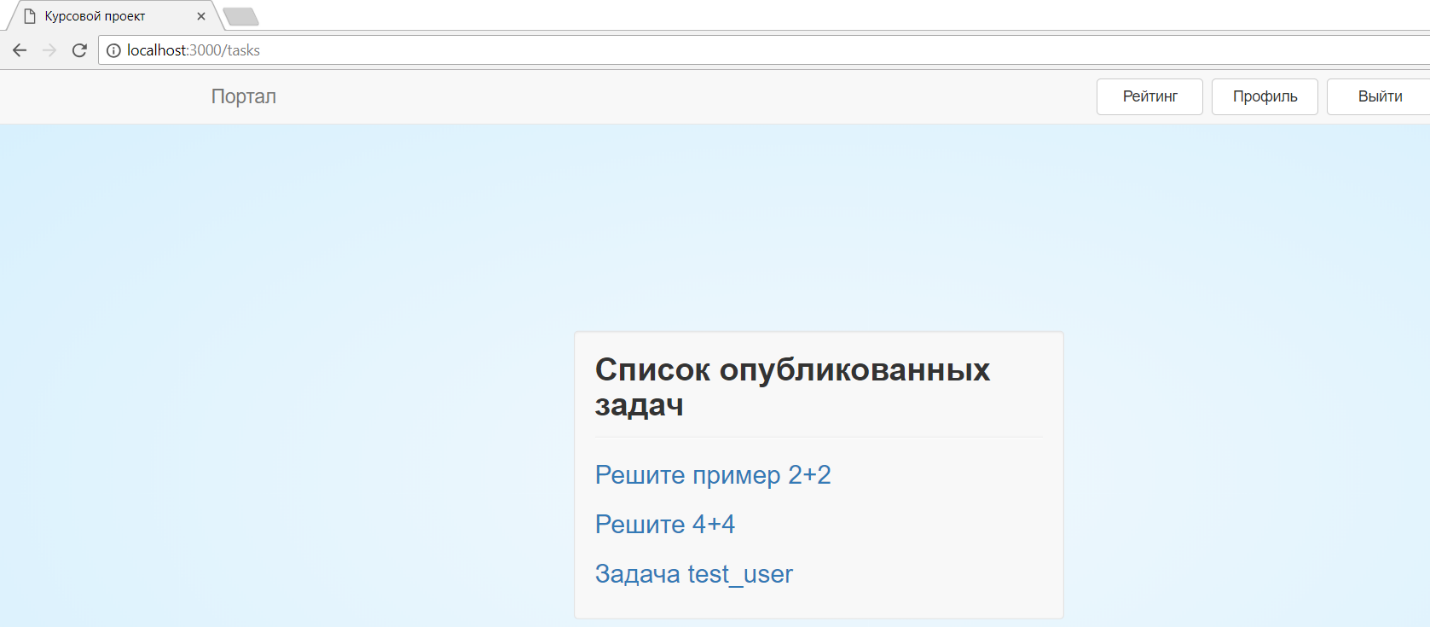
**});**

**});**

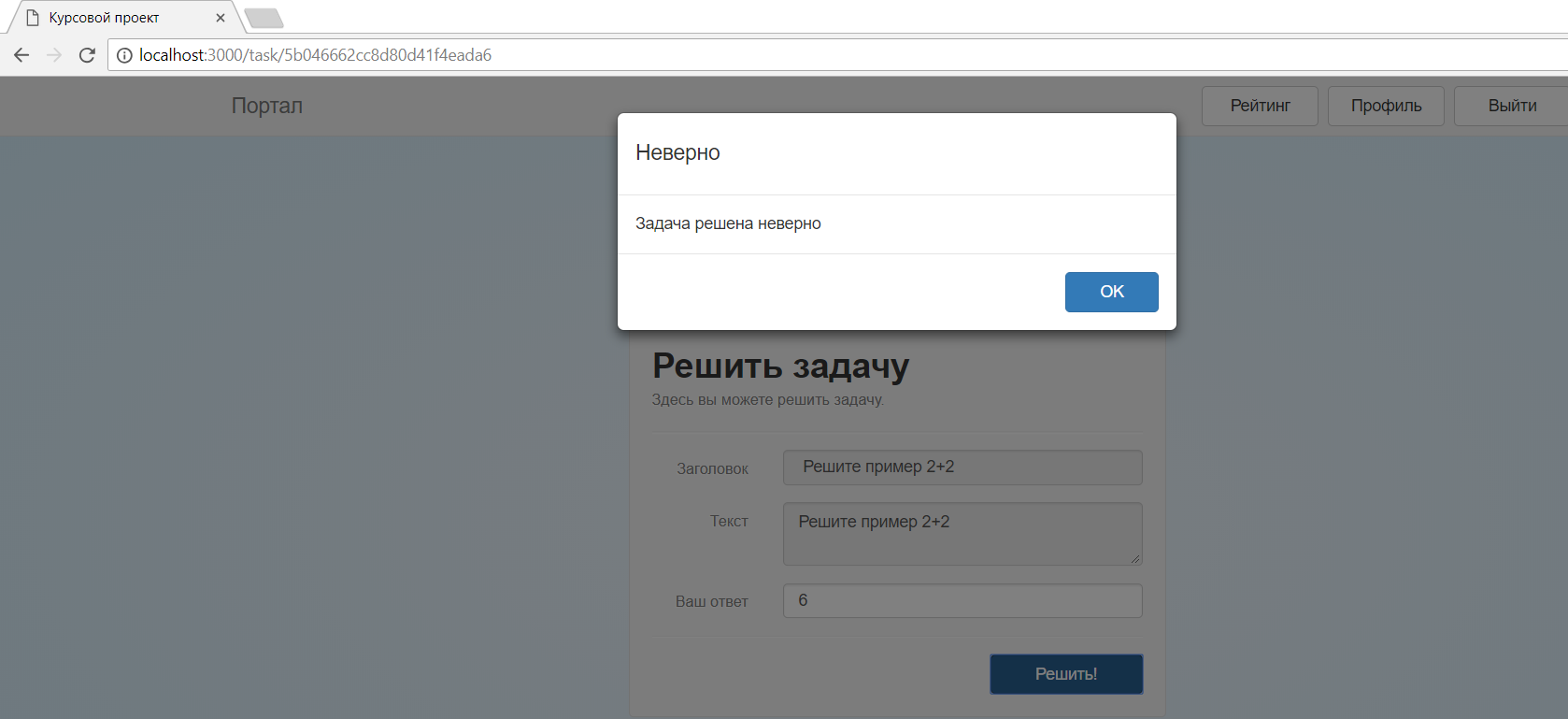
Как видно, мы достаём неоходимы нам параметры из тела запроса и сохраняем объект в коллекцию базы данных **solved\_tasks**, используя метод **addNewTask()**.

Далее создадим ещё одного пользователя **test\_user**, добавим задачу с заголовком **Задача test\_user** и попытаемся решить задачу, опубликованную пользователем **angelina.**

После проделанных действий, перейдём на страницу списка задач

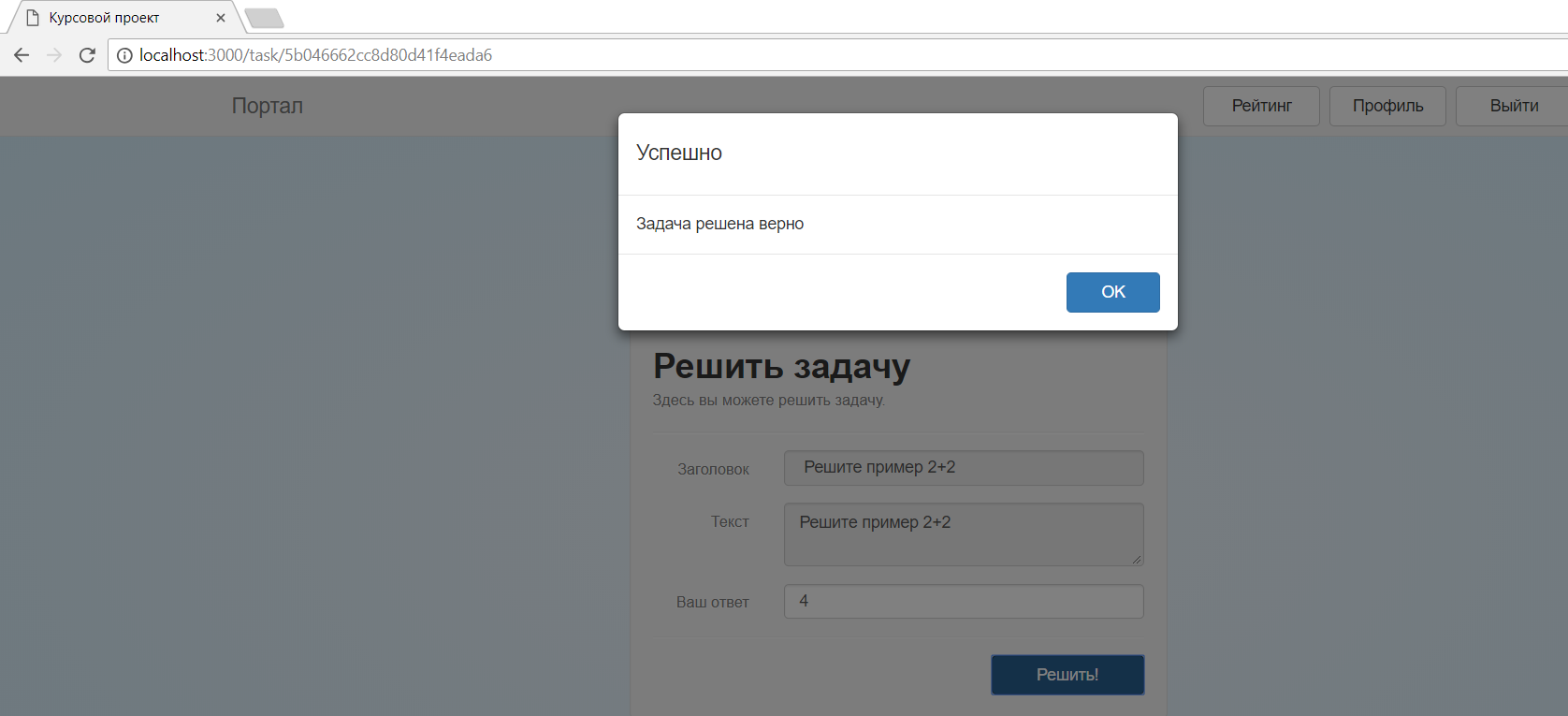


Ка видно, здесть присутсвуют задачи добавленные пользователями **angelina** и **test\_user.** Для начала попробуем решить задачу неверно

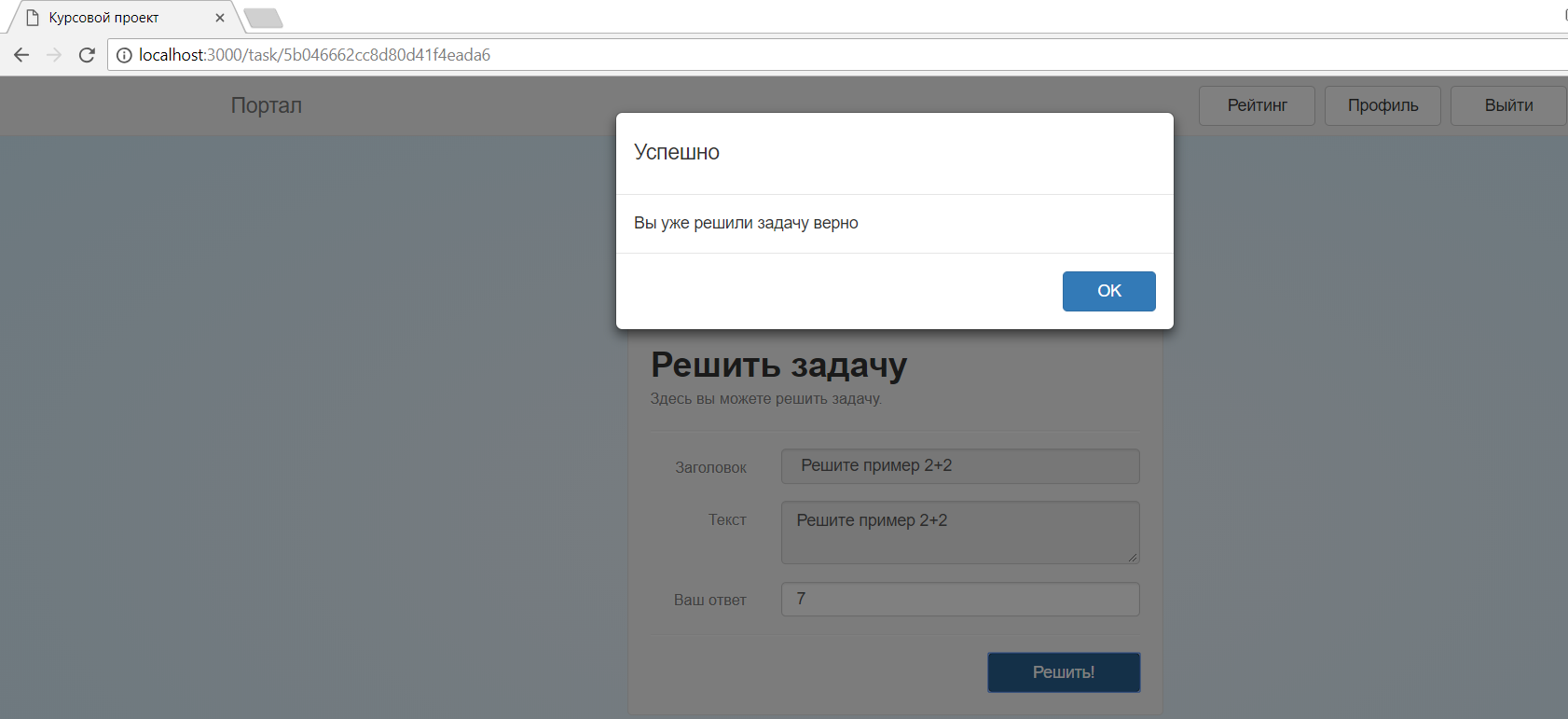


Если пользователь вводит неверный ответ, то всплывает модальное окно, которое горит, что задача решена неверно, как показано выше.

Теперь дадим правильный ответ на задачу



Попытаемся решить задачу ещё раз



Как видно, сообщение говорит, что пользователь уже решил задачу. Код, который содержит логику по обработку решения задач:

**app.post('/task/:id', function(req, res){**

**solvedTaskManager.findByIdAndUser(req.params.id, req.session.user.user, function(solved) {**

**if(solved.length > 0) {**

**res.status(200).send('already-solved')**

**} else {**

**checkCorrectAnswer(req, res)**

**}**

**})**

**});**

**var checkCorrectAnswer = function(req, res) {**

**taskManager.findById(req.params.id, function(e, foundTask){**

**if(foundTask.correct\_answer == req.body['answer']) {**

**solvedTaskManager.addNewTask({**

**\_id : req.params.id,**

**user : req.session.user.user**

**}, function(e) {**

**if (e){**

**res.status(400).send(e);**

**} else{**

**res.status(200).send('correct')**

**}**

**})**

**} else {**

**res.status(200).send('wrong')**

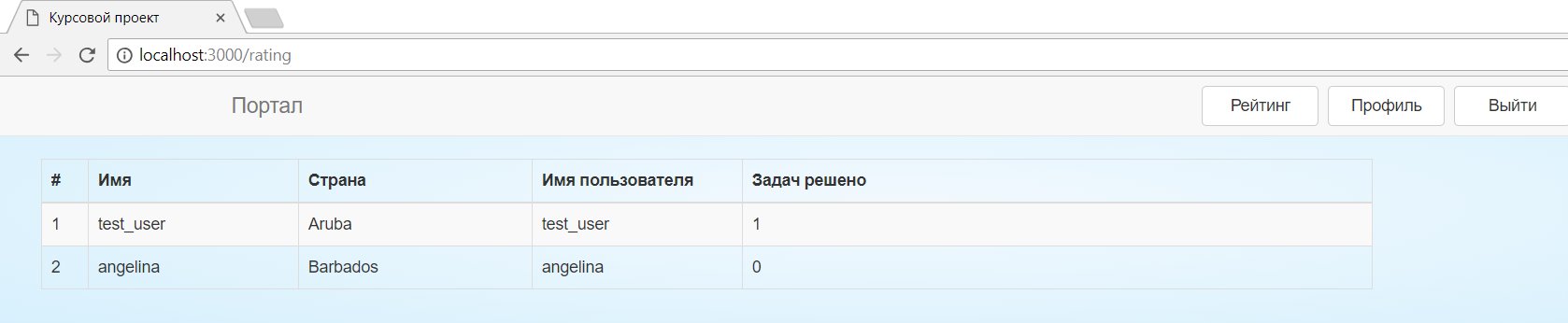
**}**

**})**

**}**

Правильно решённые задачи сохраняются в коллекцию **solved\_tasks**.

На основании количества решённых задач мы можем построить рейтинг пользователей. В нашем случае он выглядет следующим образом



**Запуск приложения**

Для запуска приложения необходимо:

1. Установить MongoDB и запустить на порту 27017
2. Установить Nodejs
3. Скачать исходный код приложения с github по ссылке

Это можно сделать следующим образом используя git

**git clone** [**https://github.com/alazovik/course\_project.git**](https://github.com/alazovik/course_project.git)

1. Перейти в командной строке папку course\_project
2. Запустить **npm install**, чтобы установить nodejs зависимости
3. Запустить сервер **node app**
4. Приложение должно запуститься на 3000 порту

Все используемы скриншоты и диаграммы, а также текст курсовой работы находятся в папке **resources**.

**Заключение**

В ходе курсового проекта были рассмотрены основные подходы к проектированию и реализации приложений, используя JavaScript. Было реализовано функционирующее веб-приложение с самостоятельно установленным веб-сервером в архитектуре Model-View-Controller с использованием технологии Ajax и AMD-концепций.

**Используемая литература**

1. Node.js Разработка серверных веб-приложений на JavaScript/ Дэвид Хэррон Изд-во: ДМК Пресс, 2012 – 146 с.
2. Изучаем JavaScript // Майкл Моррисон — СПб.: Питер, 2012. — С.608
3. <https://www.wikipedia.org/>
4. <https://nodejs.org/en/>
5. <https://www.mongodb.com/>