сессия и ввод данных

В этом проекте вы продолжите работу над предыдущим проектом, добавив возможность входа в систему, добавления комментариев к фотографиям и загрузки новых фотографий. Обратите внимание, что эти новые функциональные дополнения потребуют изменения как внешнего интерфейса (фронт-приложение), так и веб-сервера и базы данных MongoDB

Установка

Как и в предыдущем проекте, вы начинаете с копирования файлов вашего каталога предыдущего проекта в новый каталог (убедитесь, что все скрытые файлы также перенесены, включая файл .jshintrc). Извлеките содержимое этого zip-файла в новый каталог. Этот zip-файл добавит несколько новых файлов. Выполняйте всю свою работу для этого проекта в этом новом каталоге.

Чтобы выполнить это задание, вам нужно получить некоторые новые node-модули и добавить их на ваш веб-сервер и в фронт приложение. Для извлечения модуля используйте команду npm install --save <module>, которая будет извлекать модуля с именем <module> в каталог node\_modules и добавлять зависимость в файл package.json. Используйте функцию require для добавления модуля в ваш веб-сервер.

Вам нужно будет извлечь следующие модули Express в модули вашего приложения:

express-session - Экспресс-сессия – для управления сессиями.

body-parser - body-parser - для разбора тела HTTP-запросов. Вы можете использовать его для разбора закодированных JSON-запросов POST-запросов, которые мы используем в нашем API-интерфейсе сервера. Например, если вы передадите запрос с телом, состоящим из объекта JSON с параметром parameter\_name, он будет отображаться в обработчике запроса Express как request.body.parameter\_name.

multer - multer - еще один экспресс-посредник Express middleware, который способен обрабатывать формы которые содержат файлы, которые нам нужны для загрузки фотографий.

В большинстве систем следующая команда должна извлекать вышеуказанные модули:

**npm install --save express-session**

**npm install --save body-parser**

**npm install --save multer**

Добавьте их в свой webServer.js, используя инструкции require:

**var session = require('express-session');**

**var bodyParser = require('body-parser');**

**var multer = require('multer');**

Наконец, вам нужно добавить дополнительные модули в приложение:

app.use(session({secret: 'secretKey', resave: false, saveUninitialized: false}));

app.use(bodyParser.json());

где «secretKey» - произвольный ключ, который вы используете для криптографической защиты cookie сессии. Мы будем использовать библиотеку multer в коде для загрузки фотографий.

Как и в предыдущей работе, на стороне браузера вам нужно будет использовать $resource для обмена моделями с веб-сервером.

Как и в предыдущем задании, мы предоставляем тест Mocha API-интерфейса сервера, чтобы вы могли имели представление что будет возвращаться от сервера чтобы разработать код front-end.

Как и в предыдущем проекте, вам нужно будет запустить экземпляр MongoDB. Запустите MongoDB, выполнив команду:

**mongod**

и загрузите набор данных приложения для фотографий, выполнив команду:

**node loadDatabase.js**

Обратите внимание, что версия этой команды, которую мы распространяем вместе с данным проектом, загружает объект пользователя поле login\_name (строчная версия их last\_name) и поля с weak-паролем. Эти свойства появятся только при запуске loadDatabase.js после добавления полей в схему (в задаче 1).

Запуск веб-сервера Node.js

После запуска базы данных вам нужно будет запустить веб-сервер. Хотя это можно сделать с помощью той же команды, что вы использовали в предыдущих заданиях (например, node webServer.js), удобнее запускать веб-сервер с помощью команды, которая **автоматически перезапустит** ее при изменении webServer.js. В противном случае вы будете тратить время на перезапуск веб-сервера после каждого внесенного вами изменения или задаться вопросом, почему ваши изменения не сработали, когда вы забыли перезапустить его.

Командой

**npm install -g nodemon**

будет установлена ​​программа с именем nodemon, которая выполняет этот автоматический перезапуск. Обратите внимание, что флаг -g устанавливает модуль глобально, чтобы вы могли ее запускать из любом месте. Запустите свой веб-сервер с помощью команды из вашего каталога:

**nodemon webServer.js**

Команда запускает веб-сервер и перезапускает его каждый раз, когда изменяется файл webServer.js. Если у вас есть ошибка (например, синтаксическая ошибка JavaScript), которая приводит к глобальной ошибке, nodemon будет ждать, пока вы не измените файл webServer.js, а затем попробуйте перезапустить его. Если вы хотите перезапустить веб-сервер без изменения файла, вы можете перезапустить его вручную, используя параметр rs в команде nodemon. Вероятно, вы захотите запустить это в своем собственном окне, чтобы вы могли видеть сообщения журнала из вашего кода Node.js.

После обновления вашего приложения для обмена фотографиями с новыми файлами из текущего проекта и запуска базы данных и веб-сервера убедитесь, что приложение все еще работает, прежде чем продолжить задание.

Задание 1. LOGIN (15 баллов)

Разработайте модуль в приложении, чтобы иметь представление о том, что пользователь вошел в систему. Если пользователь вошел в систему, на панели инструментов должно быть указано небольшое сообщение «Привет <firstname>», где <firstname> - это первое имя зарегистрированного пользователя. Панель инструментов также должна содержать кнопку, отображающую «Выход», которая выведет пользователя из системы.

Если пользователь не вошел в систему, на панели инструментов должно отображаться сообщение «Пожалуйста, войдите в систему», и основная форма вашего приложения должна отобразить новую view с именем login-register.

Представление login-register (компоненты / login-register / login-registerTemplate.html, components / login-register / login-register.css и компоненты / login-register / login-registerController.js) должен предоставить пользователю возможность для входа в систему и, как часть проблемы № 4 ниже, зарегистрироваться как новый пользователь. Все попытки перехода к другому представлению (например, глубокие ссылки) должны привести к тому, что пользователь будет перенаправлен на представление login-register, если ни один пользователь не зарегистрировался. (См. раздел подсказок, если вы не знаете, как это реализовать.)

Кроме того, список пользователей слева не должен заполняться, если текущий пользователь не зарегистрирован. (См. раздел ниже о модификации серверного кода, чтобы вернуть статус 401 (неавторизованный)).

Когда пользователь успешно вошел, представление должно переключиться на отображение сведений пользователя. Если произошла ошибка входа (например, не найден ни один пользователь с именем login\_name), представление должно выводить соответствующее сообщение об ошибке и дать возможность повторить попытку.

Расширьте реализацию вашего бэкэнда, чтобы реализовать поддержку пользователей. При этом вы должны изменить как схему базы данных, так и API веб-сервера.

Расширьте схему Mongoose для таблицы с пользователями, чтобы добавить новое имя свойства login\_name. Это свойство представляет собой строку, содержащую идентификатор, который пользователь вводит при входе в систему.

Измените API веб-сервера для поддержки 2 новых вызовов REST API для входа и выхода пользователя. Как и в предыдущем задании, мы будем использовать HTTP-запросы с JSON-закодированными телами для передачи данных модели. API использует запросы POST для:

* **/admin/login** - Предоставляет запрос на вход. Запрос POST, в виде JSON, должен содержать свойство login\_name (пока без пароля) и возвращать информацию, необходимую вашему приложению для входа в систему. Статус HTTP 400 (неудачный запрос) должен быть возвращен, если вход в систему не выполнен (например, login\_name не является допустимой учетной записью). Доступ к параметру в элементе запроса осуществляется с помощью request.body.parameter\_name. Обратите внимание, что обработчик этого запроса должен гарантировать, что существует пользователь с данным именем login\_name. Если это так, он хранит некоторую информацию в сессии Express, где он может быть проверен другими обработчиками запросов, которые должны знать, вошел ли пользователь в систему.
* **/admin/logout** - Запрос POST с пустым телом на этот URL-адрес осуществит выход из системы пользователем, очистив информацию, хранящуюся в сеансе. HTTP-статус 400 (неверный запрос) должен быть возвращен пользователю, который в настоящее время не зарегистрирован.

В рамках обновления веб-сервера для обработки входа / выхода вам необходимо обновить все запросы (кроме / admin / login и / admin / logout), чтобы отклонить запрос со статусом 401 (неавторизованный), если в сессии нет информации о текущем пользователе.

Задание 2. Добавление комментариев

После того, как вы добавили авторизацию, следующим шагом будет возможность добавить комментариев к фотографиям. В view с фотографией, где вы показываете комментарии к фотографии, добавьте возможность для зарегистрированного пользователя добавить комментарий к фотографии. Для этой функции необходимо создать UI (например, всплывающее диалоговое окно, поле ввода и т.д.). Должно быть очевидно, как его использовать к какой фотографии комментарий. View фотографии и ее комментарии должны быть немедленно обновлены, чтобы отразить недавно добавленный комментарий.

Для поддержки этого расширьте API веб-сервера со следующим HTTP POST API:

/commentsOfPhoto/:photo\_id – добавляет комментарий к фотограции, чей идентификатор - photo\_id. Тело запросов POST должно JSON-объектом с единственным свойством **comment**, содержащим текст комментария. Объект комментария, должен содержать идентификатор зарегистрированного пользователя и время создания комментария. Ваша реализация должна отклонять любые пустые комментарии со статусом 400 (неверный запрос).

Задание 3. Загрузка фотографий

Разрешить пользователям добавлять новые фотографии. Когда пользователь вошел в систему, на главной панели инструментов должна быть кнопка с надписью «Добавить фотографию», которая позволяет текущему зарегистрированному пользователю загружать фотографию в приложение. Мы предоставим вам пример загрузки файлов с помощью HTML ниже.

Расширьте API веб-сервера, чтобы поддерживать POST-запросы по URL-адресу:

/photos/new - Загрузить фотографию для текущего пользователя. Тело запроса POST должно быть файлом (см. ниже). Загруженные файлы должны быть помещены в каталог изображений под уникальным именем, которое вы создали. Уникальное имя вместе с данными создания и зарегистрированным идентификатором пользователя должно быть помещено в новый созданный объект Photo. Статус ответа 400 должен быть возвращен, если в запросе POST нет файла.

Дополненя (5 баллов)

Теперь, когда вы пишете код, который обновляет базу данных MongoDB, ошибки в вашем коде могут повредить содержимое базы данных. Перезагрузка программы loadDatabase.js приведет к сбросу базы данных в чистое состояние. Тесты предполагают, что вы сделали это перед каждым прогоном.

Задание 1

Это задание требует, чтобы вы получили зарегистрированного пользователя как в front приложении, так и на веб-сервере.

Предполагая, что вы создали свой login-register компонент для входа, добавленный в ваше приложение для фотографий, и настроили ngRoute на использование его в пути «/login-register», полезный фрагмент кода для обработки случая когда пользователь не авторизован:

$rootScope.$on( "$routeChangeStart", function(event, next, current) {

if (noOneIsLoggedIn()) {

// no logged user, redirect to /login-register unless already there

if (next.templateUrl !== "components/login-register/login-registerTemplate.html") {

$location.path("/login-register");

}

}

});

В контроллеры необходимо добавить зависимость на $rootScope. Очевидно, вам нужно будет заменить noOneIsLoggedIn() тем, что вы используете для отслеживания состояния входа в приложение.

На стороне Angular приложения вам нужно будет изменить свои контроллеры, чтобы внедрить все пакеты, которые вам нужны для решения. Вам нужно добавить пакет как в список зависимостей, так и в список аргументов функции scope. Например, если я хочу изменить MainController в mainController.js, чтобы использовать $rootScope, $location, $http и $resource, я обновляю определение:

....controller('MainController', ['$scope', '$rootScope', '$location', '$http', '$resource',

function ($scope, $rootScope, $location, $http, $resource) {

Задание 2

В этом задании вы будете повторно использовать большую часть того, что вам нужно было выяснить для задания 1. Одно отличие - комментарии не являются отдельными объектами в нашей схеме. Они встроены в объект фотографии, поэтому вам необходимо выполнить обновление объекта, а не создание объекта.

Задание 3

Загрузка файлов затруднена в фреймворках JavaScript, так как браузеры не хотят, чтобы код JavaScript мог читать произвольные файлы. Браузеры предоставляют интерфейс, позволяющий пользователю выбирать файл, а затем отправлять его содержимое на веб-сервер. Мы можем использовать этот механизм, чтобы наше фотоприложение могло загружать фотографии.

Строкой

**<input type="file" accept="image/\*" onchange="angular.element(this).scope().inputFileNameChanged(this)">**

В браузер будет добавлена уродливая кнопка с надписью «Выбрать файл», которую пользователь может нажать, чтобы выбрать локальный файл.

Когда пользователь выбирает файл, функция inputFileNameChanged из текущего $scope будет вызываться с элементом DOM. Элемент DOM будет иметь указатель на выбранный файл в DOM.

Когда у нас есть выбранный файл в DOM, мы можем добавить его в форму DOM и отправить его в POST-запрос на веб-сервер. Вот пример для этого:

var selectedPhotoFile; // Holds the last file selected by the user

// Called on file selection - we simply save a reference to the file in selectedPhotoFile

$scope.inputFileNameChanged = function (element) {

selectedPhotoFile = element.files[0];

};

// Has the user selected a file?

$scope.inputFileNameSelected = function () {

return !!selectedPhotoFile;

};

// Upload the photo file selected by the user using a post request to the URL /photos/new

$scope.uploadPhoto = function () {

if (!$scope.inputFileNameSelected()) {

console.error("uploadPhoto called will no selected file");

return;

}

console.log('fileSubmitted', selectedPhotoFile);

// Create a DOM form and add the file to it under the name uploadedphoto

var domForm = new FormData();

domForm.append('uploadedphoto', selectedPhotoFile);

// Using $http to POST the form

$http.post('/photos/new', domForm, {

transformRequest: angular.identity,

headers: {'Content-Type': undefined}

}).then(function successCallback(response){

// The photo was successfully uploaded. XXX - Do whatever you want on success.

}, function errorCallback(response){

// Couldn't upload the photo. XXX - Do whatever you want on failure.

console.error('ERROR uploading photo', response);

});

};

Модуль экспресс, использующий body-parser, не может обрабатывать запрос POST с формой, содержащей файл, но может использовать дополнительную библиотеку с именем **multer** для извлечения файла из запроса.

var processFormBody = multer({storage: multer.memoryStorage()}).single('uploadedphoto');

processFormBody это функция, которую мы можем использовать в нашем обработчике запросов post для /photos/new.

processFormBody будет смотреть поля с именем «uploadedphoto» и вытаскивать файл из него, и записывать в свойство с именем file в объект запроса. Следующий код дает вам представление о том, как вызвать его в обработчике запросов post:

processFormBody(request, response, function (err) {

if (err || !request.file) {

// XXX - Insert error handling code here.

return;

}

// request.file has the following properties of interest

// fieldname - Should be 'uploadedphoto' since that is what we sent

// originalname: - The name of the file the user uploaded

// mimetype: - The mimetype of the image (e.g. 'image/jpeg', 'image/png')

// buffer: - A node Buffer containing the contents of the file

// size: - The size of the file in bytes

// XXX - Do some validation here.

// We need to create the file in the directory "images" under an unique name. We make

// the original file name unique by adding a unique prefix with a timestamp.

var timestamp = new Date().valueOf();

var filename = 'U' + String(timestamp) + request.file.originalname;

fs.writeFile("./images/" + filename, request.file.buffer, function (err) {

// XXX - Once you have the file written into your images directory under the name

// filename you can create the Photo object in the database

});

});

Вышеупомянутый фрагмент кода использует пакет Node.js **fs**, поэтому вам нужно будет добавить зависимость на него в свой webServer.js с помощью:

var fs = require("fs");