APPSERVER и DATABASE

Установка

**Установите последнюю "Long Term Support (LTS)" версию Node.js.**   
Она может быть установлена отсюда - <https://nodejs.org/en/download>.   
Проверить что версии установлены можно вызвав команды

node -v

и

npm -v

которые должны вывести версии установленных программ node и npm.

**Установите последнюю версию MongoDB**

Сайт: <https://www.mongodb.org/downloads#production>.

Если не указать обратного, MongoDB будет сохранять базу данных в каталоге C:\data\db

Проверьте что директория существует и даны права на запись перед тем как запускать проект.

Чтобы проверить что все БД установлена выполните команду

mongod

Это должно вывести несколько сообщений и одно из них примерно такое "waiting for connections on port 27017".

Эта команда должна быть выполнена в отдельной консоли.

Взаимодействовать с базой данных можно вызвав команду

mongo

В качестве старта возьмите свой предыдущий проект и скопируйте в новую папку.

В нее же распакуйте содержимое прилагаемого архива. Помимо новых файлов, будут обновлены файлы package.json, webServer.js, .jshintrc, и index.html ыв

После того как файлы скопированы выполните команду, чтобы установить все доп модули:

npm install

Если MongoDB server запущен необходимо загрузить данные для приложения выполнив команду

node loadDatabase.js

Эта команда загружает фейковые данные из прошлых работ в базу данных. Это действие инициализирует базу и его нужно сделать **только один раз**

Мы используем фреймворк [MongooseJS](http://mongoosejs.com/) Object Definition Language (ODL) для определения схемы данных. Файлы со схемами хранятся в папке schema:

* schema/user.js – Коллекция объектов user
* schema/photo.js – Коллекция объектов Photo + Comment
* schema/schemaInfo.js – Коллекция SchemaInfo collection содержащая объект с описанием версии текущей схемы.

Эти файлы используется скриптом loadDatabase.js в момент создания базы и скриптом webServer.js когда с базой данных работает ваше приложение.

Обратите внимание на схемы данных – модели немного отличаются от того что использовалось ранее в работах.

После того как база данных развернута, необходимо стартовать веб сервер.

node webServer.js

Обратите внимание, что сервер надо перестартовывать после каждого изменения серверного кода.

После того как все выполните, убедитесь что все работает так, как работало ранее.

Задание 1. Подключение БД(15 баллов)

The webServer.js we give you in this project is like the Project #5 webServer.js in that the app's model fetching routes use the magic cs142models rather than a database. Your job is to convert all the routes to use the MongoDB database. There should be no accesses to cs142models in your code and your app should work without the line:

var cs142models = require('./modelData/photoApp.js').cs142models;

in webServer.js. Note that any console.log statements in webServer.js will print to the terminal rather than the browser.

#### **Web Server API**

Сервер должен возвращать JSON данные по GET запросам по специфическим адресам. Спецификация путей, которую вы должны реализовать следующая:

* **/test** – Возвращает схему (/test/info) и кол-во объекто в базе (/test/count). Этот интерфейс создан для тестов и в качестве примера того, как получать информацию из БД. Метод реализован и здесь менять ничего не нужно.
* **/user/list** – Возвращает список пользователей в базе, для навигационного меню слева. Поскольку объем выгружаемой информации может быть большой, необходимо возвращать только то, что нужно для этого меню (\_id, first\_name, last\_name). Это метод заменяет метод userListModel().
* **/user/:id** – Возвращает детальную информацию о пользователе по ID. Должен возвращать то, что нужно для этого представления (\_id, first\_name, last\_name, location, description, occupation) и заменяет вызов метода userModel(). Если пользовател не найден или передано неверное ID должен возвращать осмысленное сообщение и HTTP статус 400.
* **/photosOfUser/:id** – Возвращает список фотографий пользователя по ID. Вызов генерирует набор данных необходимых для view с фотографиями, включая в себя фотографии пользователя и все комментарии к ним. По фотографиям возвращается объект (\_id, user\_id, comments, file\_name, date\_time) и объект - комментарии содержит такие свойства (comment, date\_time, \_id, user) и минимум информации о пользователе user  (\_id, first\_name, last\_name). Это заменяет вызов метода photoOfUserModel() call. Если id передан неверный то возвращается статус 400 и осмысленное сообщение.

Ваши GET запросы должны возвращать именно то, что возвращали методы в предыдущих работах, когда вы обращались к статическому json файлу. Все что нужно для представлений должно генерироваться за один вызов метода FetchModel. Все выборки и компоновки данных должны быть реализованы внутри него. Изменение схемы не допускается.

Реализация этих обработчиков запросов требует взаимодействия с двумя разными «образными» объектами данных. Система Mongoose возвращает модели из объектов, хранящихся в MongoDB, в то время как сам запрос должен возвращать модели данных, необходимые для представлений Photo App. Обратите внимание, что модели Mongoose формируются по схеме базы данных, а модели для front-end задаются потребностями представлений пользовательского интерфейса и немного отличаются. Поэтому перед отправкой на клиент необходима дополнительная пост-обработка тех данных которые получены из БД.

Необходимо соблюдать осторожность при выполнении этой обработки, поскольку модели, возвращаемые Mongoose, являются объектами JavaScript, но для них выполняется специальная обработка, поэтому любые изменения, которые не соответствуют заявленной схеме, игнорируются. Это означает, что просто обновление модели Mongoose, для передачи во front-end не cработают должным образом. Один из способов обойти это - создать копию объекта модели Mongoose. Простой способ сделать копию - перевести модель в JSON и вернуться к объектам JavaScript. Следующий фрагмент кода выполняет клонирование этого объекта:

JSON.parse(JSON.stringify(modelObject));

путем преобразования modelObject в строку JSON, а затем преобразования его обратно в объект JavaScript, на этот раз без методов и специальной обработки, выполненных на моделях Mongoose.

Задание 2. ПоДключить библиотеку $resource (10 баллов)

Для следующей работы, где вы будете реализовывать полноценное REST API, необходимо подключить и использовать библиотеку $resource вместо вашего метода FetchModel. Все методы FetchModel должны быть удалены

Необходимо подключить Angular модуль ngResource для для доступа к библиотеке resource. Документация по ngResource ([here](https://docs.angularjs.org/api/ngResource)) содержит все необходимые для этого шаги..

Несколько рекомендаций:

* Для загрузки библиотеки выполните npm из папки с проектом. Эта команда загрузит ngResource в папку node\_modules. Использование опции  ---save модифицирует файл package.json и добавит туда эту зависимость:

npm install angular-resource --save

* Файл с библиотекой будет лежать в node\_modules/angular-resource/angular-resource.js. Добавьте ссылку на этот файл(тег <script src…) в photo-share.html.
* Поскольку мы больше не используем наследование из области основного контроллера, чтобы сделать FetchModel доступным для контроллеров компонентов, нам нужно явно ввести $resource в определение главного модуля (в mainController.js) и во все контроллеры, которые должны загружать модели. Для получения подробной информации о том, как это сделать, см. документацию по [dependency injection](https://docs.angularjs.org/guide/di).
* Поскольку $resource является модулем Angular, он интегрирован в digest цикл, поэтому хаки, которые нам нужны для XMLHttpRequest на основе DOM (например, $scope. $Apply), должны быть удалены.

бонус. стиль (5 баллов)

Начисляется, если js код чистый, комментированный и JSHint  не возвращает предупреждения