Проект RADOMS: программные компоненты серверной части

Грибков К. В., студент кафедры информатики СПбГУ,
greenkirillv@gmail.com

Хайдаршин А. М., студент кафедры информатики СПбГУ,
khaydarshin.a.m@gmail.com
Суворова А.В., лаборатория теоретических и междисциплинарных проблем информатики, СПИИРАН, suvalv@mail.ru
Тулупьев А.Л., кафедра информатики СПбГУ, лаборатория теоретических и междисциплинарных проблем информатики, СПИИРАН, alexander.tulupyev@gmail.com

Аннотация

Описана схема БД профиля пользователя. Обеспечена поддержка сессий с помощью библиотек connect-mongo и session. Реализована аутентификация пользователей различными способами. Выполнена настройка маршрутов для регистрации и авторизации.

Введение

Научным работникам, исследователям и иным творческим деятелям необходимо управлять сведениями о своих результатах интеллектуальной деятельности в сфере исследований и разработок. И если с малым количеством публикаций возможно работать вручную, то при большом их количестве структурировать информацию становится сложно.

Целью настоящей работы являлась имплементация схем БД для профиля пользователя, также обеспечение аутентификации и сессионности пользователей; подзадачей являлось нахождение оптимального инструментария для разработки быстродействующей системы.

Используемые технологии

Среди множества технологий и языков программирования для написания серверной части, был выбран язык программирования JavaScript и фреймворк node.js[9]. Указанный фреймворк написан на C++, JavaScript и Си. Код программ node.js пишется на языке

JavaScript. Эта платформа имеет удобный сервис по установке пакетов — npm[10]. Указанный сервис удобен тем, что он интегрирован во многие среды разработки, например, в JetBrains WebStrom[5], и не требует дополнительной настройки. Установка так называемого "пакета" производится при помощи команды:

npm install <packagename>

В качестве http-сервера выступает фреймворк express.js[2]. Он так же написан на платформе node.js.

Для авторизации используется пакет passport.js[11]. Одно из преимуществ passport перед другими аналогами — это наличие большого количества стратегий авторизации, включая стратегии авторизации через социальные сети.

В качестве СУБД используется MongoDB[6]. Это документо-ориентированная СУБД, не требующая описания схемы таблиц, классифицируется как NoSQL база данных. Вместо традиционной реляционной структуры базы данных MongoDB использует JSON-подобные документы с динамическими схемами, из-за этого интеграция в определенных видах приложениях происходит проще и быстрее. Для работы с этой СУБД в node.js используется пакет mongoose.

Использование MongoDB

Как говорилось выше, в качестве базы данных используется документно-ориентированная MongoDB. В node.js для работы с этой базой используется пакет mongoose.

Схема профиля пользователя

Модель профиля пользователя описывается с помощью JSON-подобной схемы[8].

Листинг 1: Схема модели профиля пользователя

```
1  var Profile = new Schema({
2      local: {
3          email: {type: String},
4          username: {type: String},
5          passwordHash: String,
6          verify: {type: Boolean, default: false},
7          token: String
8      },
9      facebook: {
```

```
10          id: String,
11          token: String,
12          email: String,
13          name: String
14      },
15      vk: {
16          id: String,
17          token: String,
18          email: String,
19          name: String
20      },
21      name: {type: String},
22      surname: {type: String},
23      organization: {type: String},
24      country: String,
25      city: String,
26      phoneNo: String,
27      role: {type: Number, default: 0},
28      registrationDate: {type: Date, default: Date.now},
29  });
30
31  module.exports.Profile = mongoose.model('Profile', Profile);
```

Как видно из схемы, большинство полей имеют строковый тип. Что бы обозначить, что поле должно быть обязательно заполненно, нужно установить настройку " required"в положение true. Для обозначения того, что поле должно быть уникальным, используется настройка " unique". Настройка " default— это значение поля по умолчанию, в случае " registrationDate— это функция " Date.now которая возвращает текущую дату.

В конце файла, на 31 строке, происходит экспорт схемы, для внешнего использования.

Методы в модели

Кроме того, у модели могут быть свои методы, например, метод для шифрования пароля и метод проверки совпадения паролей:

Листинг 2: Методы профиля пользователя

```
1  Profile.methods.generateHash = function(password){
2      return bcrypt.hashSync(password, bcrypt.genSaltSync(32),
           null);
3  };
4
5  Profile.methods.checkPassword = function (password) {
6      return bcrypt.compareSync(password, this.local.
           passwordHash);
```

```
7 };
```

Пароль шифруется с помощью пакета bcrypt[1].

Виртуальные поля в модели

У модели так же могут быть виртуальные поля. Виртуальное поле — это поле которое само по себе не хранится в базе данных. Значение такого поля берется из значений других полей базы данных. При установки значения у виртуального поля, также устанавливаются значения других полей. Например виртуальное поле " password":

Листинг 3: Виртуальное поле password профиля пользователя

```
1 Profile.virtual('password')
2     .set(function (password) {
3         this.passwordHash = bcrypt.hashSync(password, bcrypt.
              genSaltSync(32), null);
4     });
```

У виртуального поля в MongoDB можно устанавливать как чтение, так и запись. В данном случае установлена только запись, которая записывает зашифрованный пароль в поле " hashedPassword". Что бы можно было прочитать виртуальное поле, нужно использовать функцию " get".

Листинг 4: Виртуальное поле userId профиля пользователя

```
1 Profile.virtual('userId')
2     .get(function () {
3         return this._id;
4     });
```

Здесь создается виртуальное поле " UserId"с возможностью чтения. Каждой записи в базе MongoDB устанавливается уникальный идентификатор, именно он возвращается при чтении виртуального поля UserId.

Сессионность

Веб-приложению необходимо иметь систему входа, реализованную с помощью сессий. Для поддержки этой возможности MongoDB и Express необходимо использовать библиотеки из менеджера пакетов npm: connect-mongo[7] и session[3] соответственно, а затем использовать данные пакеты.

Листинг 5: Объявление необходимых переменных и подключение сессий к приложению

```
1  var MongoStore   = require('connect-mongo')(session);
2  var session      = require('express-session');
3  app.use(session({
4        secret: 'yourkey',
5        resave: true,
6        saveUninitialized: true,
7        store: new MongoStore({mongooseConnection: mongoose})
8     }));
```

Переменная secret используется для генерации хэша, resave отвечает за возможность перезаписи сессии при каком-либо изменении, saveUninitialized позволяет сохранять неинициаллизированные сессии (только что созданные, и еще не измененные), store отвечает за хранилище сессий (в данном случае сессии хранятся в MongoDB).

Реализованная таким образом сессионность позволит запоминать пользователя, что избавит от необходимости повторной авторизации, например, после закрытия браузера.

Как будет показано далее, можно устанавливать максимальный срок действия текущей сессии в переменной maxAge, по истечении которого будет необходимо заново проходить авторизацию.

Стратегия аутентификации

Для интеграции сервера с базой данных пользователей используется промежуточное программное обеспечение на платформе Node.js — Passport.js. Данная надстройка позволяет проводить авторизацию с помощью различных сервисов, а также её можно использовать для создания сценария авторизации с помощью логина и пароля пользователя.

Введем переменные для обозначения стратегий аутентификации: для доступа с помощью социальных сетей Facebook или VK и для доступа с помощью пары email-пароль.

Листинг 6: Объявление необходимых переменных

```
1  app.use(passport.initialize());
2  app.use(passport.session());
3  var User = require('../models/profile').Profile;
4  var FacebookStrategy = require('passport-facebook').Strategy
       ;
5  var VkStrategy = require('passport-vkontakte').Strategy;
6  var LocalStrategy = require('passport-local').Strategy;
```

В приложении данные, использующиеся для аутентификации пользователя, будут передаваться только во время авторизации. Если пользователь существует, то информация о нем сохраняется в сессию, а идентификатор сессии, сохраняется в cookies браузера.

Каждый следующий запрос будет содержать в себе сохраненные cookies, благодаря чему passport сможет опознать пользователя и извлечь его данные из текущей сессии. Для этого необходимо использовать функции serializeUser() и deserializeUser() для начала и окончания сессии соответственно.

Листинг 7: Функции для начала и окончания пользовательской сессии

```
1  passport.serializeUser(function (user, done) {
2        done(null, user.id);
3     });
4
5  passport.deserializeUser(function (id, done) {
6        User.findById(id, function (err, user) {
7           done(err, user);
8        });
9     });
```

Стратегия регистрации

По умолчанию, LocalStrategy принимает в качестве входных данных некоторый объект с опциями и функцию, отвечающую за последующую обработку входных данных, которая принимает 3 параметра: username, password, done. Если не указывать опции, то стратегия будет искать данные в полях, которые называются username и password.

Поскольку для системы RADOMS необходима регистрация с использованием email, то необходимо передать данные сведения нашей стратегии. Это происходит с помощью переопределения usernameField. В качестве дополнительной опции укажем булевой переменной passReqToCallback истинное значение, которое позволит нам пользоваться возможностями обратного вызова.

В обрабатывающую функцию необходимо добавить четвертый аргумент req, который будет использоваться для функций обратного вызова. Параметр done принимает вторым аргументом объект пользователя, если он существует.

При регистрации выполняется проверка на существование пользователя с таким электронным адресом: если пользователя с таким электронным адресом найдено не было, выполняется создание объекта пользователя.

Листинг 8: Стратегия регистрации

```
passport.use('local-signup', new LocalStrategy({
        usernameField: 'email',
        passwordField: 'password',
        passReqToCallback: true
    },
    function (req, email, password, done) {
        process.nextTick(function () {
            if (req.body.emailUsername == undefined ||
                req.body.emailUsername === true) {
                User.findOne({'local.email': email}, function
                     (err, user) {
                    if (err) {
                        return done(err);
                    }
                    if (user) {
                        return done(err, {message: 'This email
                            has already used'});
                    } else {
                        var newUser = new User();
                        newUser.local.email = email;
                        newUser.local.password = newUser.
                            generateHash(password);
                        newUser.name = req.body.name;
                        newUser.surname = req.body.surname;
                        newUser.organization = req.body.
                            organization;
                        newUser.country = req.body.country;
                        newUser.city = req.body.city;
                        newUser.phoneNo = req.body.phoneNo;
                        newUser.save(function (err) {
                            if (err) {
                                return done(err);
                            }
                            req.session.cookie.maxAge = 31 * 24
                                * 60 * 60 * 1000;
                            return done(null, newUser);
                        });
                    }
                });
            }
            else {
                User.findOne({'local.username': email},
                     function (err, user) {
                    if (err) {
                        return done(err);
                    }
                    if (user) {
                        return done(err, {message: 'This email
                            has already used'});
```

```
43                      } else {
44                          var newUser = new User();
45                          newUser.local.username = email;
46                          newUser.local.password = newUser.
                                generateHash(password);
47                          newUser.save(function (err) {
48                              if (err) {
49                                  return done(err);
50                              }
51                              req.session.cookie.maxAge = 31 * 24
                                    * 60 * 60 * 1000;
52                              return done(null, newUser);
53                          });
54                      }
55                  });
56              }
57          })
58      }));
```

Стратегия авторизации

Стратегия авторизации во многом схожа со стратегией регистрации пользователя.

Проводится проверка на существование пользователя с введенным логином в базе данных, затем проводится проверка на корректность введенного пароля для данного пользователя. В случае отсутствия пользователя в базе данных, выводится сообщение с соответствующей ошибкой.

Листинг 9: Стратегия авторизации

```
1  passport.use('local-login', new LocalStrategy({
2          usernameField: 'email',
3          passwordField: 'password',
4          passReqToCallback: true
5      },
6      function (req, email, password, done) {
7          if (passLimiter.check(req)) {
8              User.findOne({
9                  $or: [
10                     {'local.email': email},
11                     {'local.username': email}
12                 ]
13             }, function (err, user) {
14                 if (err) {
15                     return done(err);
16                 }
17                 if (!user) {
```

```
18                  return done({message: 'Incorrect email'},
                        null);
19              }
20              if (!user.checkPassword(password)) {
21                  return done({message: passLimiter.add(req)
                        ||
22                  'Incorrect password'}, null);
23              }
24              if (!user.local.verify) {
25                  req.res.json({message: 'Confirm your
                        registration ' +
26                  'before entering!', 'type': 'info'});
27                  return;
28              }
29              req.session.cookie.maxAge = req.body.
                    rememberMe ? 31 * 24
30              * 60 * 60 * 1000 : 24 * 60 * 60 * 1000;
31              return done(null, user);
32          });
33      }
34      else {
35          return done({message: "You are blocked"});
36      }
37  }));
```

Аутентификация через социальные сети

Стратегия аутентификации через социальные сети Facebook[4] и VK[14] ничем не отличается, поэтому рассмотрим её на примере Facebook.

Чтобы сервисы Facebook могли идентифицировать приложение, с которого будет посылаться запрос на авторизацию, его нужно зарегистрировать. Это делается по ссылке https://developers.facebook.com/. Необходимо указать название приложения, его тип, а также дополнительные поля, в зависимости от выбранных ранее параметров.

После регистрации открывается страница редактирования приложения. Во вкладке "Настройки" находятся "Идентификатор приложения" и "Секрет приложения", оба этих поля понадобятся для запросов на сервера Facebook.

Настройка логики авторизации

Настройка логики авторизации осуществляется с помощью т.н. стратегий. В случае "Facebook"нужная стратегия находится в пакете

passport-facebook (для других социальных сетей, есть другие пакеты: VK — passport-vkontakte; twitter[13] — passport-twitter и т.д.).

Логика авторизации, которая приведена ниже, состоит в том, что пользователь может регистрироваться только через электронную почту и пароль, а потом, в профиле, может прикрепить аккаунт социальной сети, чтобы в дальнейшем через неё авторизоваться.

Листинг 10: Стратегия авторизации с помощью социальной сети Facebook

```
1  passport.use(new FacebookStrategy({
2         clientID: configAuth.facebookAuth.clientID,
3         clientSecret: configAuth.facebookAuth.clientSecret,
4         callbackURL: configAuth.facebookAuth.callbackURL,
5         passReqToCallback: true
6     }, function (req, token, refreshToken, profile, done) {
7         process.nextTick(function () {
8             if (!req.user) {
9                 User.findOne({'facebook.id': profile.id},
                      function (err, user) {
10                    if (err)
11                        return done(err);
12                    if (user) {
13                        return done(null, user);
14                    }
15                    else
16                        done(null, false, {message: "User not
                              found"});
17                });
18            }
19            else {
20                var user = req.user;
21                user.facebook.id = profile.id;
22                user.facebook.token = token;
23                user.facebook.name = profile.displayName;
24                user.save(function (err) {
25                    if (err)
26                        throw err;
27                    req.session.cookie.maxAge = 31 * 24 * 60 * 60
                          * 1000;
28                    return done(null, user);
29                });
30            }
31        });
32
33    }));
```

Нужно передать поля, необходимые для авторизации:

ClientID — это "Идентификатор приложения";

ClientSecret — это "Секрет приложения";

callbackURL — это URL, на который будет возвращаться ответ "Facebook";

passReqToCallback — присваивая этому полю значение true, мы сообщаем passport'у, что в теле функции понадобится непосредственно сам запрос клиента. Если бы этого поля не было или его значение было равно false, то первым аргументом функции был бы token, а не req.

Чтобы сообщить passport'у об успешности или неуспешности авторизации, нужно возвратить в теле стратегии функцию done с соответствующими аргументами:

- если возникла ошибка, то первым аргументом будет эта ошибка;
- если пользователь успешно авторизован, то первым аргументом будет null, а вторым — найденная запись пользователя;
- если пользователь не идентифицирован, то первым аргументом будет null, а вторым — false.

Далее следует сама логика авторизации. Сначала определяется авторизован ли был пользователь в момент запроса. Это делается с помощью поля req.user, оно будет не определено, если пользователь не авторизован.

1. Если пользователь не авторизован, то его, соответственно, нужно авторизовать. Что бы это сделать, пользователя нужно идентифицировать, делается это с помощью функции User.findOne, которая ищет запись в базе данных по полю facebook.id. Далее мы сообщаем об успешной или неуспешной идентификации или ошибке.
2. Если пользователь авторизован, то к текущему профилю прикрепляются данные для авторизации по социальной сети "Facebook". Далее сообщается об успешности или неуспешности сохранения изменений в профиле пользователя.

Настройка маршрутов

Стратегии регистрации пользователя и авторизации были описаны в предыдущем разделе, осталось подготовить их к использованию.

При регистрации или авторизации пользователя используется метод POST, предназначенный для запросов, при которых сервер принимает данные из внешних источников в теле сообщения запроса. При обращении к данному методу во время авторизации происходит перенаправление на страницу профиля пользователя в успешном случае, а при обращении во время регистрации происходит перенаправление на api для верификации пользователя.

Листинг 11: Регистрация и авторизация пользователя

```
app.post('/api/auth/local/login', passport.authenticate('
    local-login', {
      successRedirect: '/profile',
      failureRedirect: '/about'
    }));

    app.post('/api/auth/local/join', passport.authenticate('
        local-signup'),
     function (req, res) {
      res.redirect('/api/auth/local/verification');
    });
```

Авторизация через социальные сети происходит следующим образом:

1. Клиент переходит на страницу /api/auth/facebook;
2. passport перенаправляет клиента с соответствующим запросом на сайт "Facebook"для авторизации;
3. После авторизации "Facebook"перенаправляет пользователя на страницу /api/auth/facebook/callback.

Листинг 12: Маршруты авторизации через социальные сети

```
app.get('/api/auth/facebook',
   passport.authenticate('facebook'));

app.get('/api/auth/facebook/callback',
       passport.authenticate('facebook', {
          successRedirect: '/success-redirect',
          failureRedirect: '/failure-redirect'
       }));

app.get('/api/unconnect/facebook', function (req, res) {
       if (!req.user) {
          res.redirect("/");
          return;
```

```
14          }
15          var user = req.user;
16          user.facebook = undefined;
17          user.save(function (err) {
18              res.redirect('/')
19          })
20      });
```

Так же выше представлен обработчик запроса страницы /api/unconnect/facebook. При переходе на эту страницу, удаляется поле user.facebook, чтобы в дальнейшем по старым данным невозможно было авторизоваться.

Заключение

В рамках настоящей работы были успешно решены поставленные задачи, связанные с созданием схемы профиля пользователя, реализацией сессионности и стратегий аутентификации, а также авторизацией пользователя в приложении RADOMS.

Все частные цели, а, в следствии, и общая цель достигнуты. Сервис доступен по адресу http://radoms.ru/[12] и находится в стадии тестирования.

Литература

[1] Bcrypt documentation. — URL: `https://www.npmjs.com/package/bcrypt-nodejs` (дата обращения: 25.05.2016).

[2] Express documentation. — URL: `http://expressjs.com/en/api.html` (дата обращения: 25.05.2016).

[3] Express-session. — URL: `https://www.npmjs.com/package/express-session` (дата обращения: 25.05.2016).

[4] Facebook. — URL: `https://www.facebook.com/` (дата обращения: 25.05.2016).

[5] JetBrains WebStorm. — URL: `https://www.jetbrains.com/webstorm/` (дата обращения: 25.05.2016).

[6] MongoDB documentation. — URL: `https://docs.mongodb.org/manual/` (дата обращения: 25.05.2016).

[7] Mongodb session store for connect. — URL: https://www.npmjs.com/package/connect-mongodb (дата обращения: 25.05.2016).

[8] Mongoose documentation. — URL: http://mongoosejs.com/docs/guide.html (дата обращения: 25.05.2016).

[9] Node.js documentation. — URL: https://nodejs.org/api/ (дата обращения: 25.05.2016).

[10] Npm. — URL: https://www.npmjs.com/ (дата обращения: 25.05.2016).

[11] Passport.js documentation. — URL: http://passportjs.org/docs/overview (дата обращения: 25.05.2016).

[12] RADOMS. — URL: http://radoms.ru/ (дата обращения: 25.05.2016).

[13] Twitter. — URL: https://twitter.com/ (дата обращения: 25.05.2016).

[14] VK. — URL: https://vk.com/ (дата обращения: 25.05.2016).