УДК 004.432.2

Янко А.С., к.т.н.,
Яковенко П.Л. студент,
Полтавський національний технічний
університет імені Юрія Кондратюка

РОЗРОБКА СИСТЕМИ РЕЄСТРАЦІЇ ТА АВТЕНТИФІКАЦІЇ ДЛЯ ВЕБ-ДОДАТКУ

***Анотація**. У даній статті описано процес розробки системи реєстрації та автентифікації для веб-додатку на основі технологій ReactJS, NodeJS, MongoDB. Дана система може бути використана в різних типах додатків, легко масштабується та має властивість до легкого розширення свого функціоналу за допомогою встановлення додаткових модулів.*

***Ключові слова:** автентифікація, реєстрація, веб-додаток, веб-програмування, база даних, інтерфейс користувача.*

Вступ

У наш час веб-сайти та веб-додатки грають досить важливу роль в повсякденному житті. Всі хто має на своєму комп’ютері або мобільному гаджеті підключення до мережі Інтернет зрозуміють ці слова. Отримувати та відправляти інформацію від одного користувача до іншого через Інтернет неможливо без взаємодії з веб-сайтами або веб-додатками. А взаємодія між користувачами неможливо без системи реєстрації та автентифікації, оскільки кожен користувач повинен мати данні для посилання та отримання даних, як і в реальному житті. Дана система дозволяє створювати так званий акаунт користувача в базі даних користувачів та надалі використовувати цей обліковий запис для роботи з веб-ресурсом.

В даній роботі буде розглянуто створення системи реєстрації та автентифікації за допомогою таких технологій як React.js, Node.js та MongoDB. Дана система буде використовуватися у веб-додатку для обміну миттєвими повідомленнями.

Принцип роботи даної системи полягає в тому, що користувач вводить свою адресу електронної пошти та пароль, після чого ці дані передаються серверу, який хешує пароль користувача для забезпечення більш безпечного зберігання даних, та записує цю інформацію в базу даних. Перед кожним циклом реєстрації сервер перевіряє базу даних на наявність такої самої електронної адреси як у користувача, що хоче зареєструватися. Якщо така адреса існує - сервер відправляє відповідь про повторну реєстрацію адреси, після чого за допомогою технології на якій будується інтерфейс користувача, в даному випадку це React.js, висвітлюється повідомлення до користувача про те, що дана адреса вже використовується, і йому потрібно ввести іншу адресу електронної пошти. Також під час реєстрації інтерфейс користувача здійснює валідацію адреси електронної пошти користувача, щоб вона задовольняла вимоги щодо структури електронної адреси. Якщо адреса введена неправильно, то висвітлюється відповідне повідомлення яке інформує користувача ввести коректну електронну адресу для подальшої реєстрації.

Процес автентифікації є більш складним, оскільки на нього покладено великі повноваження – він надає доступ до інформації користувача. Після успішної реєстрації користувач має можливість за допомогою відомих адреси електронної пошти та паролю, увійти до акаунта користувача. Після введення даних інтерфейс користувача відправляє запит до сервера з електронною адресою та паролем користувача. Сервер у свою чергу, отримавши ці дані, починає пошук в базі даних користувачів та перевіряю, чи існує запис з такою адресою електронної пошти. Знайшовши запис, сервер кодує пароль, який отримано з запиту про автентифікацію та порівнює його з тим, що записано в базі даних для відповідного користувача. Якщо вони збігаються - сервер генерує успішну відповідь та відправляє дані користувача разом з JSON Web

токеном, який генерується для забезпечення доступу до різних Routes веб-додатку саме цього користувача, а не будь-якого іншого. Отримавши позитивну відповідь від сервера, інтерфейс користувача змінює Rout на такий, що йому відображається компонент, який відповідає за його обліковий запис. Дані отримані разом з відповіддю сервера, використовуються для відображення персональних даних на сторінці користувача. JSON Web Token записується в локальне сховище браузера для подальшого використання при звертанні до сервера від імені користувача. Надалі для доступу до інших компонентів додатку та отримання різного роду інформації, додаток в заголовку кожного запиту також відправляє JSON Web Token, який на серверній частині перевіряється, на основі перевірки сервер вирішує чи надавати доступ користувачу до запрошених даних або дій, чи ні.

Створення бази даних та стратегії для роботи бібліотеки Passport.js

Розглянемо процес створення даної системи в деталях, на реальному прикладі системи реєстрації та автентифікації.

Спочатку створюється схему користувача в базі даних, схема надає можливість базі даних зрозуміти який саме вид матиме об’єкт користувача.

```
4   const userSchema = mongoose.Schema({
5       local: {
6           email: String,
7           password: String,
8           thumbnail: String,
9           status: String,
10          name: String,
11          lastName: String,
12          logged: Boolean,
13          position: String,
14          location: String,
15          team: String
16      }
17  });
```

Рис. 1.Схема користувача в базі даних

Схема користувача буде мати вигляд об'єкта (Рис.1). Об'єкт матиме ключ local – який відповідає процесу реєстрації користувача за адресою електронної пошти та паролем. Таке розділення викликане великим різноманіттям варіантів реєстрації іншими способами. Такими як реєстрації за допомогою соціальних мереж, або інших веб-сервісів, наприклад - Google+, GitHub та інші. Таке розділення дозволить в майбутньому додати інші типи реєстрації для даного додатку.

Наступним кроком буде конфігурування шляху підключення до бази даних, а також встановлення секретних ключів для JSON Web Token.

```
module.exports.dbaseConfig = {
    url: 'mongodb://admin:admin.mlab.com:13713/social'
}
```

Рис. 2.Конфігурування адреси бази даних

Для підключення сервера до бази даних, задається її адреса на віддаленому сервісові mLab (Рис.2).

```
module.exports.config = {
    jwtsecret: 'jwtsecret',
    jwtSession: {
        session: false
    }
}
```

Рис. 3.Конфігурування секретів для JWT

Конфігурування секретів потрібне для правильної роботи токенів надалі, саме ці секрети будуть використовуватися при кодуванні токена (Рис.3). Також задано, що сесії використовуватися не будуть.

Після цього потрібно конфігурувати стратегію для роботи JWT.

Стратегія – це механізм за допомогою якого бібліотека Passport.js верифікує вхідні запити на сервер та пропускає цей запит через стратегію, яка вирішує, чи має цей запит потрібні дані для доступу до інформації. Створено

стратегію яка буде отримувати payload, що являється розшифрованим JSON Web токеном та за електронною адресою користувача здійснює пошук в базі даних (Рис.4). Якщо такого користувача знайдено – система відправляє відповідь з об'єктом, що містить знайдену інформацію. Якщо користувача не знайдено або під час перевірки виникла якась помилка – система відправляє відповідний статус для інтерфейсу користувача.

```
13      const strategy = new JwtStrategy(opts, (payload, done) => {
14          User.findOne({'local.email': payload.email}, (err, user) => {
15              if (err) {
16                  return done(err, false);
17              }
18              if (user) {
19                  return done(null, user);
20              } else {
21                  return done(null, false, 'no such user');
22              }
23          });
24      });
25      passport.use(strategy);
```

Рис. 4.Створення стратегії для роботи JWT

Наступним кроком є створення інтерфейсу користувача, який буде отримувати введені користувачем данні та передавати їх серверу за допомогою AJAX запитів.

Створення інтерфейсу користувача

Наступним кроком є створення інтерфейсу користувача, який буде отримувати введені користувачем данні та передавати їх серверу за допомогою AJAX запитів.

Задається регулярний вираз який потім порівнюється з переданим адресом електронної пошти (Рис.5). Якщо адреса відповідає вимогам – відбувається створення AJAX запиту. Якщо ж адреса не відповідає вимогам – задається помилка в електронній адресі, та користувач бачить це на екрані.

```
14    submitHandler (e) {
15        e.preventDefault();
16        this.setState({wrongEmail: false});
17        const emailRegExp = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))
18        if (this.email.value.match(emailRegExp) && this.password.value.length > 0) {
19            this.props.actions.signUp(this.email.value, this.password.value);
20        } else {
21            this.setState({
22                wrongEmail: true
23            })
24        }
25    }
```

Рис. 5.AJAX запит до сервера для реєстрації користувача

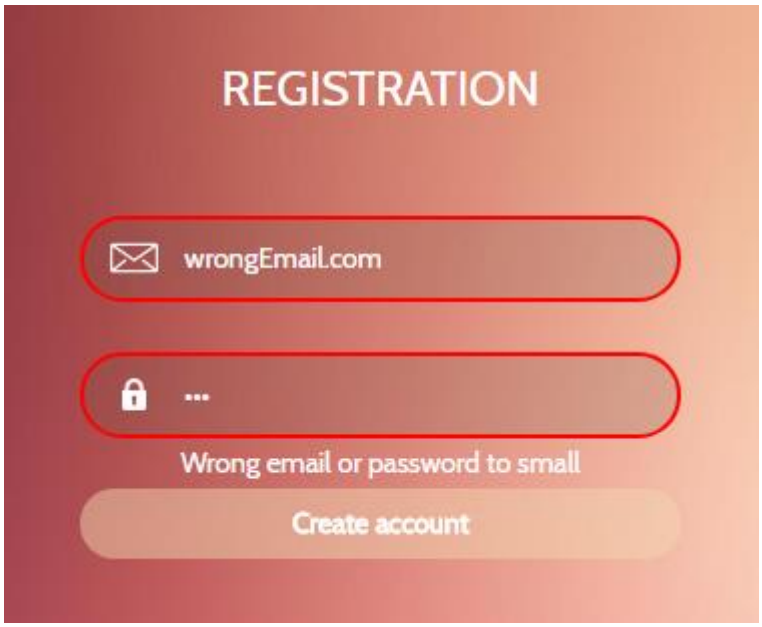

Рис. 6.Реакція інтерфейсу користувача на неправильну електронну адресу

Користувач буде сповіщений про те, що він ввів некоректну адресу електронної пошти або некоректний пароль (Рис.6).

Запит буде відповідати за передачу адреси електронної пошти та паролю до сервера. Для створення AJAX запитів використовується бібліотека Axios.

Відправляється AJAX запит до сервера, в тілі запиту передається електронна адреса та пароль користувача, за допомогою бібліотеки Axios, після отримання відповіді від сервера, вирішується якими будуть наступні дії (Рис.7). Якщо відповідь є позитивною – інтерфейс користувача відобразить кореневий компонент додатку, з якого користувач зможе увійти до свого акаунта.

```
11  axios.post(`${rootUrl}/sign-up`, {email, password}).then((returnedData) => {
12      if (returnedData.data.success) {
13          browserHistory.push('/');
14      } else if (returnedData.data.message == 'Email in use') {
15          dispatch({
16              type: types.SET_ERROR,
17              errorType: 'emailInUse'
18          });
19      }
20  });
```

Рис. 7.AJAX запит до сервера для реєстрації користувача

Якщо ж відповідь серверу, що електронна адреса використовується – за допомогою Redux диспатчиться дія, що задає тип помилки – електронна адреса у використанні. Після зміни типа помилки в глобальному сховищі, інтерфейс користувача показує відповідну реакцію.

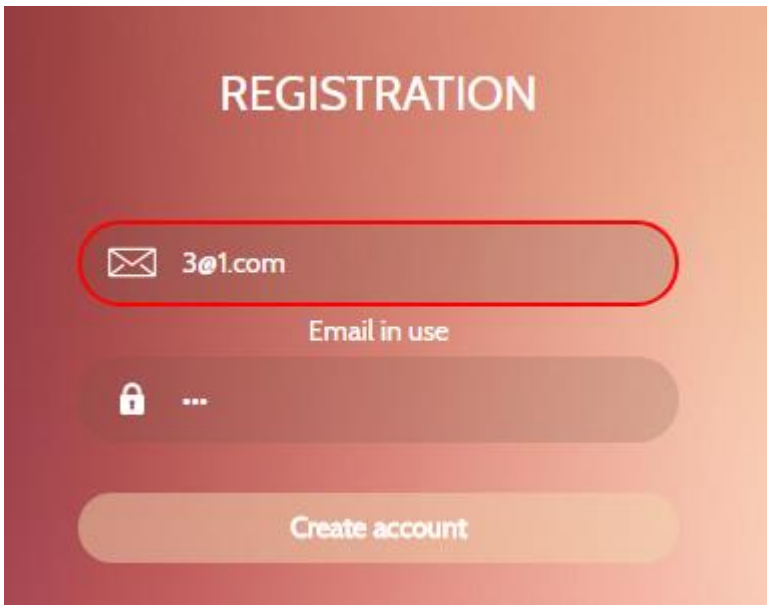

Рис. 8.Реакція інтерфейсу на реєстрацію по вже існуючій адресі

Інтерфейс показує користувачеві – дана адреса електронної пошти у використанні (Рис.8).

Якщо ж адреса не використовується та пароль задовольняє усі умови – відбувається процес створення та збереження об’єкту користувача у базі даних.

Цей процес є досить важливим через те, що формує скелет бази даних, створюючи реєстр усих користувачів веб-додатку.

```
app.post('/sign-up', (req, res) => {
    const {email, password} = req.body;
    if (!email && !password) res.json({message: 'provideData'})
    User.findOne({ 'local.email': email }, (err, user) => {
        if (err) console.log('error while searching for registred user')
        if (user) res.json({message: 'Email in use'})
        else {
            var newUser = new User();
            newUser.local.email = email;
            newUser.local.password = newUser.generateHash(password);
            newUser.local.logged = false;
            newUser.local.location = 'Not specified',
            newUser.local.team = 'Not specified'
            io.emit('changed user data');
            newUser.save(function(err) {
                if (err) throw err;
                res.json({success: true})
            });
        }
    });
});
```

Рис. 9.Створення об’єкту користувача у базі даних

Об’єкт користувача створюється тільки тоді, коли адреси електронної пошти, яка була передана в запитові, не існує в базі даних. На рис.9 ініціалізується новий користувач за допомогою уже створеної завчасно стратегії. Далі вказуються всі поля в об’єктові, а також генерується подія в Socket для сповіщення про те, що створено нового користувача, отже усим клієнтам потрібно повторно завантажити список усих користувачів для отримання коректної інформації. Після цього відбувається збереження користувача, якщо під час збереження з’являється якась помилка - виконання збереження буде перервано. В іншому випадку, сервер відправляє статус, що збереження успішне, після чого інтерфейс користувача відображає кореневий компонент.

Система автентифікації

Наступним кроком є автентифікація користувача, що намагається увійти до свого профілю. Інтерфейс користувача буде мати майже той самий вигляд, адже для автентифікації потрібні ті самі речі – адреса електронної пошти та пароль. Після введення цих даних, користувач натискає кнопку – Увійти. При натисканні кнопки інтерфейс користувача відправляє AJAX запит до сервера з тілом запиту, що містить адресу та пароль.

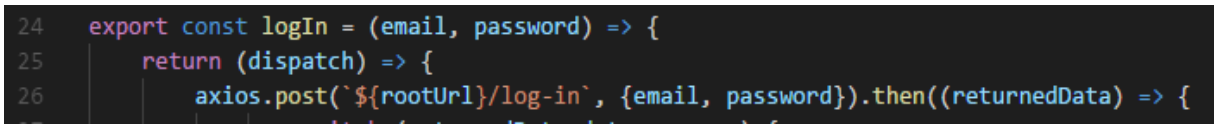

Рис. 10.AJAX запит для автентифікації користувача

AJAX запит передає вміст форми, в яку користувач вводить свої дані (Рис.10). Після посилання AJAX запиту, йде очікування на дані які будуть отримані від сервера. В даному випадку йде очікування на декілька статусів. У випадку автентифікації вони можуть бути: користувача з такою адресою електронної пошти не існує, користувач існує, але пароль не є вірним або ж все успішно і користувач ввів дані вірно.

Action виконує різні дії в залежності від відповіді сервера (Рис.11). Наприклад, якщо користувача не знайдено – сервер відправляє статус ‘userNotFound’, після чого йде dispatch дії яка задає помилку, що користувач не знайдений. Після зміни статусу помилки інтерфейс користувача автоматично виводить це на екран. Аналогічні дії відбуваються з неправильно введеним паролем. Але якщо користувач ввів дані правильно – сервер відправляє JSON Web Token та електронну адресу користувача, що увійшов до системи. Після отримання позитивної відповіді від сервера, action задає для локального сховища браузера адресу та токен, які будуть використані надалі при роботі. Також йде dispatch дії, що задає в глобальному сховищі Redux об’єкт користувача, для його швидкого відображення в інтерфейсі веб-додатку.

```
switch (returnedData.data.message) {
    case 'userNotFound':
        return dispatch({
            type: types.SET_ERROR,
            errorType: 'userNotFound'
        });
    case 'wrongPassword':
        return dispatch({
            type: types.SET_ERROR,
            errorType: 'wrongPassword'
        });
    case 'loggedSuccessfully':
        localStorage.setItem('token', returnedData.data.token)
        localStorage.setItem('email', returnedData.data.user.local.email)
        dispatch({
            type: types.SET_USER,
            user: returnedData.data.user
        });
        browserHistory.push('/panel');
}
```

Рис. 11.Перебір можливих варіантів відповіді від сервера

В свою чергу серверна частина при автентифікації відповідає за знаходження користувача, якщо такий існує – хешування переданого паролю та порівняння його з записаним в базі даних. Якщо йде повний збіг - сервер відправляє статус успіху, JWT та об’єкт користувача.

Відбувається перевірка на наявність користувача (Рис.12). Якщо його знайдено - йде хешування переданого паролю та порівняння його з тим, що існує. Якщо пароль не збігається – сервер передає відповідний статус на інтерфейс користувача. Якщо інформація збігається - сервер змінює статус користувача на онлайн, генерує подію – користувач змінив дані, для обновлення даних на екранах усих користувачів хто онлайн. Це зроблено для відображення тих, хто увійшов у веб-додаток без перезавантаження сторінки. Результатом дії є переданий JWT, об’єкт користувача та статус успіху.

Надалі для доступу до захищених шляхів додатку буде використовуватися JSON Web Token, який буде передаватися в заголовкові кожного запиту та відповідатиме за те, що саме цей користувач хоче отримати чи змінити дані відносно свого облікового запису.

```
app.post('/log-in', (req, res) => {
    const {email, password} = req.body;
    User.findOne({'local.email': email}, (err, user) => {
        if (err) throw (err);
        if (!user) {
            res.json({message: 'userNotFound'});
        } else {
            bcrypt.compare(password, user.local.password, (err, valid) => {
                if (!valid) {
                    res.json({message: 'wrongPassword'});
                } else {
                    user.local.logged = true;
                    io.emit('changed user data');
                    user.save((err) => {
                        if (err) throw (err);
                        const token = jwt.sign({email: user.local.email}, jwtsecret);
                        res.json({token, user, message: 'loggedSuccessfully'});
                    });
                }
            });
        }
    });
});
```

Рис. 12.Перевірка адреси та паролю під час автентифікації користувача

```
    app.use((req, res, next) => {
            auth.authenticate(req, res, next);
        }
    );
```

Рис. 13.Встановлення middleware для сервера

Встановлюється так зване middleware, яке буде надавати доступ до захищених шляхів додатку тільки тоді коли запит буде проходити перевірку за допомогою стратегії Passport.js (Рис.13).

Висновок

Отже, у даній роботі було розглянуто механізм створення та функціонування системи реєстрації та автентифікації користувача у веб-додатку. Даний спосіб є сучасним методом, тому що використовує технології, які є актуальними при створенні веб-сайтів та веб-додатків на сьогодні. Спосіб не залежить від бази даних або технології для створення інтерфейсу

користувача, тому що головна його частина є серверною, тому тільки зміна алгоритму та мови написання серверної частини може потягти за собою зміни в реалізації механізмів реєстрації та автентифікації. Даний спосіб легко масштабується та може бути доповнений великою кількістю різноманітних способів, таких як – реєстрація через соціальні мережі чи інші веб-сервіси. Також варто зазначити, що стек технології, що використовувався під час створення даного рішення є дуже прогресивним напрямом у Full Stack програмуванні, тому що він базується на мові JavaScript, яка є однією з найпоширеніших високорівневих мов програмування у світі.

Посилання

1. passportjs.org – Офіційна документація бібліотеки Passport.js [Електронний ресурс] – Режим доступу до ресурсу: http://www.passportjs.org.

2. expressjs.com – Офіційна документація до фреймворку для Node.js [Електронний ресурс] – Режим доступу до ресурсу: http://expressjs.com/.

3. Ethan Brown Web Development with Node & Express / Ethan Brown // – 2014. – С. 329.

Authors:
Yanko Alina Sergeevna, Yakovenko Pavel Leonidovich

DEVELOPMENT OF REGISTRATION AND AUTHENTICATION SYSTEMS FOR A WEB APPLICATION

Abstract. This article describes the process of developing a registration and authentication systems for a Web application based on ReactJS, NodeJS, MongoDB. This system can be used in different types of applications, easily scalable and has the ability to easily extend its functionality by installing additional modules.

Keywords: authentication, registration, web application, web programming, database, user interface.

Авторы:
Янко Алина Сергеевна, Яковенко Павел Леонидович

РАЗРАБОТКА СИСТЕМЫ РЕГИСТРАЦИИ И АУТЕНТИФИКАЦИИ ДЛЯ ВЕБ-ПРИЛОЖЕНИЯ

Аннотация. В даной статье описывается процесс разработки систем регистрации и аутентификации для веб-приложения на основе ReactJS, NodeJS, MongoDB. Эта система может использоваться в различных типах приложений, легко масштабируема и имеет возможность легко расширять функциональность, устанавливая дополнительные модули.

Ключевые слова: аутентификация, регистрация, веб-приложение, веб-программирование, база данных, интерфейс пользователя.