

Title: Двухфакторная аутентификация (2FA) через SMS в приложении

Ключевые слова: двухфакторная аутентификация, 2fa, информационная безопасность, мобильные приложения, разработка мобильных приложений

пожелания по картинке: [референс](#), [два](#), [три](#), можно добавить другие какие-то элементы, кроме устаревшей техники

Хабы: Блог компании МТС, Информационная безопасность, Программирование, Софт

Как сделать двухфакторную аутентификацию через SMS в своём приложении

Добавление двухфакторной аутентификации в веб-приложение — самый простой способ уменьшить количество спама и мошенничества на своём сайте, обеспечив при этом безопасность пользователя. Двухфакторная аутентификация защищает от фишинга, атак методом социальной инженерии и перебора паролей. Даже если пароль пользователя скомпрометирован, этого недостаточно, чтобы предоставить злоумышленнику доступ к аккаунту в вашем приложении: без утверждения второго фактора пароль сам по себе бесполезен.

Меня зовут Анастасия Иванова, я технический писатель [MTC Exolve](#). В этой статье я расскажу, как можно реализовать двухфакторную аутентификацию в веб-приложении на NodeJS, и объясню, как отправлять одноразовый код через [SMS API](#), используя сервис MTC Exolve.

<cut />

Что нам понадобится

- [Аккаунт разработчика в MTC Exolve](#)
- API-ключ приложения в аккаунте разработчика. Инструкции о том, как создать приложение и найти его API-ключ, вы можете найти в статьях «[Создание приложения](#)» и «[API-ключ приложения](#)»
- Купленный номер Exolve, с которого будем отправлять SMS. Инструкцию о том, как купить номер, вы можете найти в статье «[Покупка номера](#)»
- [Node.js](#) и следующие библиотеки:
 - [express](#) (создание сервера веб-приложения)
 - [body-parser](#) (парсинг тела входящего HTTP-запроса с клиентской части веб-приложения с номером телефона пользователя)
 - [axios](#) (отправка HTTP-запроса в Exolve API для отправки SMS с одноразовым кодом пользователю)
 - [nunjucks](#) (шаблонизатор для JavaScript — понадобится для передачи данных с сервера на клиентскую часть приложения)

Установка библиотек

Установите библиотеки, необходимые для работы нашего Node.js-приложения. Если у вас ещё не установлен Node.js, вы можете скачать его с [официального сайта](#). Вместе с ним установится npm — пакетный менеджер для скачивания внешних библиотек.

Создайте проект для приложения. Для этого выполните команду инициализации в консоли:

```
npm init
```

После выполнения команды введите название приложения, описание, имя автора и другие данные. После ввода и подтверждения всех данных будет создан package.json-файл. Он будет содержать информацию о приложении и зависимостях — сторонних библиотеках для его работы.

Установите библиотеки, которые понадобятся далее. Выполните следующую команду в консоли:

```
npm i -s express body-parser axios nunjucks
```

Она установит библиотеки, указанные в пункте «[Что нам понадобится](#)». После успешной установки в package.json-файле должны появиться зависимости от библиотек. Пример файла:

```
{
  "name": "2fa",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Anastasia Ivanova",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.4.0",
    "body-parser": "^1.20.2",
    "express": "^4.18.2",
    "nunjucks": "^3.2.4"
  }
}
```

Основа Express.js сервера приложения

Создайте основу Express приложения:

1. Подключите установленные библиотеки и укажите, что приложение их использует.
2. Укажите порт приложения.
3. Укажите ответ приложения на клиентский GET-запрос к главной странице (используйте простой текстовый “Hello World” в основе).
4. Укажите, что приложение слушает запросы на указанном порте, и сделайте вывод соответствующего сообщения в консоль.

Для этого создайте index.js-файл в корне проекта и добавьте туда следующий код:

```
// Подключение библиотек
const app = require('express')(); // наше приложение app работает на
базе Express
const bodyParser = require("body-parser");
const axios = require('axios');
const nunjucks = require('nunjucks');

// Входящие HTTP-запросы обрабатываются библиотекой body-parser
app.use(bodyParser.json());
app.use(bodyParser.urlencoded( {extended: false} ))

// Порт, на котором работает наше приложение
const port = 3001;

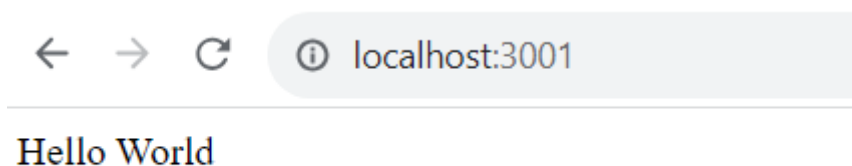
// Ответ на клиентский запрос к главной странице приложения
app.get('/', (req, res) => {
  res.send("Hello World");
});

// Приложение будет слушать запросы на указанном выше порте
app.listen(port, () => {
  console.log(`App listening at http://localhost:${port}`)
});
```

Сохраните изменения и запустите приложение. Для этого выполните в консоли команду:

```
node index.js
```

В консоли появится сообщение о том, что приложение работает на порте 3001. Перейдите по ссылке <http://localhost:3001/> и получите ответ от сервера “Hello World”:



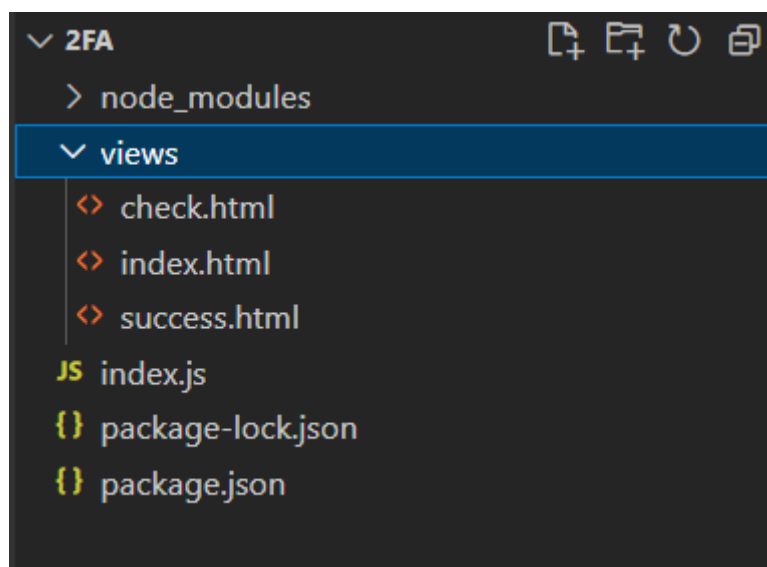
Клиентская часть приложения

Клиентская часть приложения будет содержать 3 страницы:

1. Главная страница, где пользователь сможет ввести свой номер телефона.
2. Страница для ввода и проверки одноразового кода.
3. Страница успешной аутентификации.

Создайте папку `views` в корне приложения, поместите туда три HTML-страницы: `index.html`, `check.html` и `success.html`.

После этого структура приложения будет такой:



Главная страница — `index.html`

На главной странице понадобится поле для ввода номера телефона пользователя и кнопка, при нажатии на которую клиентская часть отправит введённые данные на сервер приложения.

Добавьте в файл `index.html` следующий код:

```

<!-- Текстовое сообщение, которое мы можем получить от сервера -->
{{message}}

<!-- Форма для ввода номера телефона и отправки POST-запроса на точку
доступа /verify -->
<form method="post" action="verify">
    <input name="phoneNumber" type="tel">
    <button>Получить код</button>
</form>

```

Страница проверки одноразового кода — check.html

Как и на главной странице, здесь понадобится форма для ввода полученного одноразового кода и кнопка отправки введённых данных на сервер.

Добавьте в файл следующий код:

```

<!-- Текстовое сообщение, которое мы получим от сервера при рендеринге
check.html -->
{{message}}

<!-- Форма для ввода кода подтверждения и отправки POST-запроса на
точку доступа приложения /check -->
<form method="post" action="check">
    <input name="code" type="Введите код подтверждения">
    <input name="phoneNumber" type="hidden" value="{{ phoneNumber }}">
<!-- Скрытое поле с номером телефона пользователя, полученное с сервера
при рендеринге -->
    <button>Отправить</button>
</form>

```

Обратите внимание, что здесь форма содержит скрытое поле с предустановленным значением "{{ phoneNumber }}". При рендеринге страницы сервер приложения подставит туда номер телефона пользователя, введённый на главной странице. Это поле понадобится для сравнения номера телефона пользователя и одноразового кода. Как это сделать, расскажу дальше в статье.

Страница успешной аутентификации — success.html

Здесь всё просто: нужно вывести сообщение об успешной аутентификации, если на предыдущей странице пользователь ввёл верный одноразовый код:

```
<!-- Текстовое сообщение, которое мы получим от сервера при рендеринге  
success.html -->  
{{message}}
```

Серверная часть приложения

Сервер приложения будет отправлять SMS-сообщения пользователям с номера телефона, купленного в вашем аккаунте Exolve. Для этого нужно отправить POST-запрос к Exolve SMS API, в котором понадобятся API-ключ и купленный номер ([инструкция по отправке SMS через Exolve](#)).

В «боевом» режиме такие данные стоит хранить в переменных окружения для безопасности. Для простоты демонстрации в нашем примере мы объявим их как константы в index.js-файле. Добавьте следующий код:

```
const url = 'https://api.exolve.ru/messaging/v1/SendSMS'; // Точка  
доступа Exolve API для отправки SMS  
const exolveNumber = '79XXXXXXXXX'; // купленный номер  
const apiKey = 'YOUR_API_KEY'; // API-ключ
```

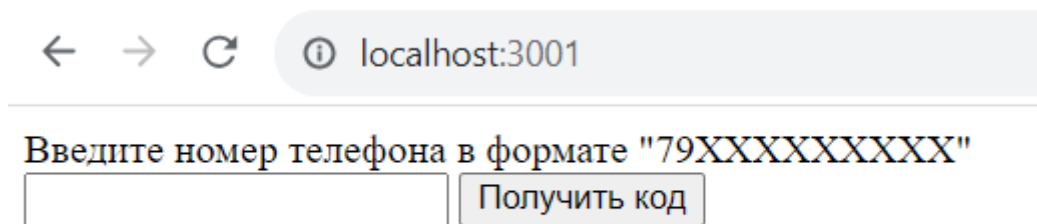
Клиентская часть приложения состоит из трёх HTML-файлов, которые находятся в папке views. Укажите шаблонизатору nunjucks, что он должен рендерить файлы из этой папки:

```
// nunjucks рендерит файлы из папки views  
nunjucks.configure('views', { express: app });
```

При запросе клиента к главной странице приложения сервер должен срендерить index.html-файл. Замените app.get-запрос из основы, созданной выше, на следующий код:

```
app.get('/', (req, res) => {  
  res.render('index.html', { message: 'Введите номер телефона в  
формате "79XXXXXXXXX"' });  
});
```

Теперь, когда вы запустите приложение и откроете <http://localhost:3001/> в браузере, вы должны увидеть сообщение, переданное сервером при рендеринге, и форму для ввода номера телефона:



← → ↻ ⓘ localhost:3001

Введите номер телефона в формате "79XXXXXXXXXX"

Когда пользователь введёт номер телефона и нажмёт на кнопку «Получить код», клиентская часть отправит POST-запрос на сервер к точке доступа /verify. При получении такого запроса сервер должен:

1. Получить номер телефона из тела запроса.
2. Сгенерировать одноразовый код.
3. Запомнить пару номер телефона + одноразовый код для последующего сравнения.
4. Отправить SMS с одноразовым кодом на номер телефона пользователя.
5. Показать ошибку, если сообщение не может быть отправлено, или страницу для ввода кода, если сообщение отправлено.

Напишем функцию для генерации случайного кода с помощью встроенных в JavaScript функций [Math.random\(\)](#) и [Math.floor\(\)](#):

```
function generateCode(min, max) {  
    return Math.floor(Math.random() * (max - min) + min);  
}
```

Нужно хранить пару номер телефона + сгенерированный код для последующей проверки соответствия. В «боевом» режиме стоит сохранять эту пару в базу данных. Для простоты будем записывать данные в объект и добавлять в массив. Как это сделать?

Создайте пустой массив users:

```
users = []; // массив для хранения пар номер телефона + одноразовый код
```

Теперь добавьте функцию для сохранения объекта с номером телефона и одноразового кода в массив:

```
function addUser (phoneNumber, code) {  
    // создаём объект на основе полученного номера телефона и  
    // сгенерированного одноразового кода  
    user = {  
        phoneNumber: phoneNumber,  
        code: code  
    }  
}
```

```

// Проверяем, есть ли в массиве users объект с указанным номером
userIndex = users.findIndex(el => el.phoneNumber == phoneNumber);

if (userIndex == -1) { // Если нет, добавляем созданный объект в массив
    users.push(user);
} else { // Если есть, заменяем одноразовый код на новый
    users[userIndex].code = code;
}
}

```

Далее напишите функцию для [отправки SMS-сообщения](#) с одноразовым кодом с помощью библиотеки axios:

```

async function sendVerificationCode(phoneNumber, code) {

    var text = "Одноразовый код: " + code;
    // Пробуем отправить SMS
    try {
        await axios({
            method: 'post',
            url: url,
            headers: {'Authorization': 'Bearer ' + apiKey},
            data: {
                number: exolveNumber,
                destination: phoneNumber.toString(),
                text: text
            }
        })
    }
    .then((response) => {
        result = response.data; // Записываем ответ от Exolve API в переменную
    });
    } catch (error) {
        return error.response.data.error // Возвращаем текст ошибки, если SMS не было отправлено
    }

    return result // Возвращаем ответ от Exolve (message_id при успешной отправке SMS)
}

```

Сделайте обработку POST-запроса клиентской части к точке доступа /verify:

```

app.post('/verify', async (req, res) => {
    const phoneNumber = req.body.phoneNumber; // номер пользователя из
тела запроса
    const code = generateCode(1000, 9999); // генерируем случайный
четырёхзначный код
    addUser (phoneNumber, code); // добавляем пару номер + код в массив
users
    const result = await sendVerificationCode(phoneNumber, code); //
отправляем SMS с одноразовым кодом
    if (result.message_id !== undefined) { // если функция отправки
кода возвращает нам message_id, рендерим страницу ввода кода
        res.render('check.html', { phoneNumber: phoneNumber, message:
'Введите код подтверждения' }); // передаём номер телефона
пользователя, который будет в скрытом поле
    } else { // в случае ошибки снова рендерим главную страницу с
сообщением об ошибке отправки
        console.log(result.details);
        res.render('index.html', { message: 'Сообщение не может быть
доставлено. Проверьте правильность введённого номера. Формат номера
"79XXXXXXXXXX"' });
    }
});

```

При успешной отправке SMS-сообщения с кодом пользователь попадёт на страницу для ввода кода. При нажатии кнопки «Отправить» клиентская часть отправит POST-запрос с данными на сервер к точке доступа /check. Напишите обработку этого запроса:

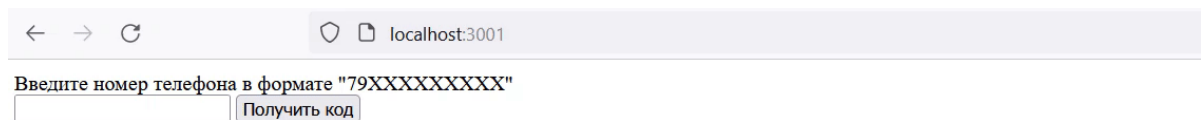
```

app.post('/check', (req, res) => {
    const phoneNumber = req.body.phoneNumber; // номер пользователя из
тела запроса (скрытое поле)
    const code = req.body.code; // введённый пользователем код
    userIndex = users.findIndex(el => el.phoneNumber == phoneNumber);
// ищем индекс объекта с номером телефона
    if (users[userIndex].code == code) { // если введённый код
совпадает с кодом в объекте, рендерим страницу успешной аутентификации
        res.render('success.html', { message: 'Вы успешно
авторизованы!' });
    } else { // Если код не совпадает, рендерим страницу ввода кода с
сообщением об ошибке
        res.render('check.html', { message: 'Неверный код подтверждения.
Введите правильный код.' });
    }
});

```

```
} ;  
}) ;
```

Запустите приложение, чтобы проверить, как всё работает:



The screenshot shows a web browser window with the address bar displaying 'localhost:3001'. Below the address bar, there is a text prompt: 'Введите номер телефона в формате "79XXXXXXXXXX"'. Underneath this prompt is a text input field and a button labeled 'Получить код' (Get code).

Таким образом, мы реализовали двухфакторную аутентификацию через отправку одноразового кода через [SMS API](#). Полный код приложения вы можете найти на [GitHub](#).

В конце статьи хотим напомнить, что у нас [в сообществе MTC Exolve проходит творческий конкурс](#) — вы можете присылать истории, связанные с профессиональным опытом в разработке. Участвуйте и получайте гарантированные призы.