**Проектная работа 14**

**Авторизация и регистрация пользователя**

**1. Добавьте email и password к схеме пользователя**

Регистрировать пользователей будем по почте и паролю. Поэтому в схему пользователя добавьте два поля: email и password. У каждого пользователя email должен быть уникальным и валидироваться на соответствие схеме электронной почты.

Для валидации воспользуйтесь модулем validator: <https://www.npmjs.com/package/validator>.

По умолчанию база возвращает поле password при запросе к модели пользователя. Пока не меняйте его — к этому вернёмся позже.

**2. Доработайте контроллер createUser**

Помимо полей name, about и avatar, он должен добавлять поля email и password. Хешируйте пароль, прежде чем сохранять его в базу.

Поля name, about и avatar сделайте необязательными. Если при создании пользователя эти поля не были указаны, то им должны присваиваться стандартные значения — для этого доработайте схему user. Подробнее [про стандартные значения вы можете прочитать здесь](https://mongoosejs.com/docs/defaults.html#declaring-defaults-in-your-schema).

Для этих полей используйте следующие стандартные значения из макета предыдущих спринтов:

* name — «Жак-Ив Кусто»;
* about — «Исследователь»;
* avatar — [ссылка](https://pictures.s3.yandex.net/resources/jacques-cousteau_1604399756.png);
* Картинка

Данные всех полей должны приходить в теле запроса.

**3. Создайте контроллер login**

В файле controllers/users.js создайте контроллер login, который получает из запроса почту и пароль и проверяет их. Если почта и пароль правильные, контроллер должен создавать JWT сроком на неделю. В пейлоуд токена следует записывать только свойство \_id, которое содержит идентификатор пользователя:

Скопировать кодJAVASCRIPT

{

\_id: "d285e3dceed844f902650f40"

}

JWT после создания должен быть отправлен клиенту. Мы рекомендуем записывать JWT в httpOnly куку. Если вам проще сделать это в теле ответа, такое решение тоже будет принято.

При неправильных почте и пароле контроллер должен вернуть ошибку 401.

**4. Создайте роут для логина и регистрации**

В app.js создайте два обработчика POST-запросов на два роута: '/signin' и '/signup':

Скопировать кодJAVASCRIPT

app.post('/signin', login);

app.post('/signup', createUser);

Передайте им соответствующие контроллеры — login и createUser. Из файла routes/users.js удалите обработчик создания пользователя — он больше не нужен.

**5. Сделайте мидлвэр для авторизации**

В файле middlewares/auth.js создайте мидлвэр для авторизации. Он должен верифицировать токен из заголовков. Если с токеном всё в порядке, мидлвэр должен добавлять пейлоуд токена в объект запроса и вызывать next:

Скопировать кодJAVASCRIPT

req.user = payload;

next();

Если с токеном что-то не так, мидлвэр должен вернуть ошибку 401.

**6. Создайте контроллер и роут для получения информации о пользователе**

Реализуйте роут:

Скопировать код

GET /users/me - возвращает информацию о текущем пользователе

**7. Защитите API авторизацией**

Защитите авторизацией все маршруты, кроме страницы регистрации и логина.

**8. Удалите хардкод**

В самостоятельном задании предыдущего спринта мы добавили такой мидлвэр, поскольку тогда вы ещё не умели реализовывать авторизацию:

Скопировать кодJAVASCRIPT

app.use((req, res, next) => {

req.user = {

\_id: '5d8b8592978f8bd833ca8133'

};

next();

});

Теперь авторизация готова, поэтому удалите этот мидлвэр.

**9. Проконтролируйте права**

У пользователя не должно быть возможности удалять карточки других пользователей.

**10. Сделайте так, чтобы API не возвращал хеш пароля**

В соответствующее поле схемы пользователя нужно добавить свойство select со значением false:

Скопировать кодJAVASCRIPT

const userSchema = new Schema({

*// ...*

password: {

type: String,

required: true,

select: false *// необходимо добавить поле select*

},

*// ...*

});

Так по умолчанию хеш пароля пользователя не будет возвращаться из базы.

Но в случае аутентификации хеш пароля нужен. Чтобы это реализовать, после вызова метода модели, нужно добавить вызов метода select, передав ему строку +password:

Скопировать кодJAVASCRIPT

User.findOne({ email }).select('+password')

.then((user) => {

*// здесь в объекте user будет хеш пароля*

});

**Примерная структура проекта**

Структура проекта должна выглядеть как-то так:



Обратите внимание: в коде нет env-файла. Мы добавим его в следующей части работы, когда загрузим проект на сервер. Секретный ключ для разработки можно хранить в коде, это не страшно.

**Последние штрихи**

**1. Реализуйте централизованную обработку ошибок**

Создайте мидлвэр для централизованной обработки ошибок. В местах возникновения ошибок больше не возвращайте их, а передавайте обработку в этот мидлвэр.

Проследите за тем, чтобы API не возвращал объект ошибки в том виде, в котором он возник. Например, такого быть не должно:

Скопировать кодJAVASCRIPT

.catch(err => res.send(err))

В случае, если на сервере возникает ошибка, которую мы не предусмотрели, возвращайте ошибку 500.

В процессе работы с регистрацией и авторизацией у вас появились новые роуты и методы в контроллерах. Не забудьте обработать ошибки и там.

Учтите также случай, когда пользователь пытается зарегистрироваться по уже существующему в базе email. Для этого вам пригодится 409 код ошибки и следующая конструкция:

Скопировать кодJAVASCRIPT

if (err.name === "MongoError" && err.code === 11000) {

*// Обработка ошибки*

}

Такую ошибку и код отдает сама база данных при попытке создать дубликат уникального поля.

Если при попытке авторизоваться присланный токен некорректен — возвращайте 401 статус-код.

**2. Валидируйте приходящие на сервер запросы**

Тела запросов к серверу должны валидироваться до передачи обработки в контроллеры. Если запрос принимает какую-то информацию в заголовках или параметрах, валидируйте и её.

API должен возвращать ошибку, если запрос не соответствует схеме, которую мы определили.

**3. Валидируйте данные на уровне схемы**

Для проверки данных поля avatar используйте регулярное выражение.

Убедитесь, что ссылка на аватар выполняет условия:

https?:\/\/(www\.)?[-a-zA-Z0-9@:%.\_\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-9()@:%\_\+.~#?&//=]\*)

1. Начинается с http:// или https://.
2. www. — это необязательная группа.
3. Путь — последовательность из цифр, латинских букв и символов -.\_~:/?#[]@!$&'()\*+,;=, указанных после названия домена и доменной зоны. На конце пути может стоять решётка #.
4. Шаблон находит url таких форматов:

Скопировать код

http://ya.ru

https://www.ya.ru

http://2-domains.ru

http://ya.ru/path/to/deep/

http://ya-ya-ya.ru

В этом задании самостоятельно разберитесь, как валидировать ссылки. Это описано в документации mongoose: <https://mongoosejs.com/docs/validation.html>.

**Заключение**

Когда всё готово — можно открывать шампанское. Вы написали все части фронтенда и бэкенда стандартного (но очень классного) приложения. Останется только соединить это вместе, а после — выложить в публичный доступ. Но об этом в следующем спринте.

Чеклист

Не забудьте проверить себя по чеклисту: <https://code.s3.yandex.net/web-developer/checklists/new-program/checklist-14/index.html>