Чеклист для самопроверки №14

Перед отправкой работы на ревью убедитесь, что она соответствует всем критериям, которые описаны в этом документе.

Обратите внимание, что этот чеклист — продолжение предыдущего. То есть нужно следовать всем пунктам из чеклиста к предыдущей работе, а также новым требованиям.

Работа отклоняется от проверки

Ревьюеры не станут проверять работу, если она соответствует хотя бы одному критерию из этого блока:

- В `README.md` не приведена ссылка на репозиторий проекта.
- Ошибки при выполнении команды `npm run start` и `npm run dev`.
- В `.gitignore` не добавлено исключение `node_modules`.
- Реализована не вся обязательная функциональность.
- Работа содержит вопросы или просьбы о помощи к ревьюеру.
- На повторных итерациях не исправлены критические замечания.

Работа принимается

Приложение корректно обрабатывает запросы по следующим роутам:

В дополнение к роутам, описанным в задании предыдущей проектной работы, необходимо реализовать следующие роуты:

- `POST /signup` — регистрация пользователя;
- `POST /signin` — авторизация пользователя;
- `GET /users/me` — возвращает информацию о текущем пользователе.

Корректно работает авторизация и регистрация пользователя:

- В схеме пользователя есть обязательные поля `email` и `password`.
- Поле `email` должно быть уникальным — есть опция `unique: true`.
- Поле `password` не ограничено в длину, так как пароль хранится в виде хеша.
- В контроллере `createUser` почта и хеш пароля записываются в базу.
- Создание пользователя в контроллере производится вызовом метода `create` модели, без дополнительной проверки на существование пользователя, так как поле `email` в схеме отмечено как уникальное.
- Поля `name`, `about` и `avatar` пользователя необязательные. Если они не заполнены в запросе, то в них подставляются стандартные значения.
- Есть контроллер `login`, который проверяет полученные в теле запроса почту и пароль.
- Если почта и пароль верные, контроллер `login` создаёт JWT, в `payload` которого записано свойство `_id` с идентификатором пользователя.
- JWT-токен выпускается на определённый срок (например, 7 дней), а не даётся бессрочно.
- В ответ на успешную авторизацию контроллер `login` возвращает клиенту созданный токен — это может быть тело ответа или заголовок `Set-Cookie`.
- Удалён хардкод `req.user` из проектной работы предыдущего спринта.
- Есть файл `middlewares/auth.js`, в нём middleware авторизации для проверки JWT.
- Роуты `/signin` и `/signup` не должны обрабатываться middleware авторизации.
- При правильном JWT авторизационный middleware добавляет в объект запроса `payload` токена и пропускает запрос дальше.
- Пользователь не может удалить карточку, которую он не создавал.

- Пользователь не может редактировать чужой профиль и менять чужой аватар.
- API не возвращает хеш пароля.
- Все роуты, кроме `/signin` и `/signup`, должны быть защищены авторизацией.

Валидация данных:

- Поле `email` пользователя валидируется на соответствие паттерну почты, можно использовать пакет validator;
- Тела запросов и, где необходимо, параметры запроса и заголовки должны валидироваться по определённым схемам с помощью celebrate;
- Если запрос не соответствует схеме, обработка не должна передаваться контроллеру, клиент получает ошибку валидации;
- В контроллере не должно быть валидации данных, если она описана с помощью celebrate;
- Поля `avatar` и `link` проверяются регулярным выражением. Шаблон находит URL таких форматов:

```
http://ya.ru
https://www.ya.ru
http://2-domains.ru
http://ya.ru/path/to/deep/
http://ya-ya-ya.ru
```

Обработка ошибок в приложении:

- Если в любом из запросов что-то идёт не так, сервер возвращает ответ с ошибкой и соответствующим ей статусом. Помимо ошибок с кодами **400**, **404** и **500,** описанных в задании предыдущей проектной работы, в запросах должны также обрабатываться следующие ошибки:
  - **401** — передан неверный логин или пароль. Также эту ошибку возвращает авторизационный middleware, если передан неверный JWT;
  - **403** — попытка удалить чужую карточку;
  - **409** — при регистрации указан email, который уже существует на сервере.
- Для ошибок созданы классы конструкторы ошибок, наследуемые от `Error`.
- Не дублируются классы конструкторы ошибок с одинаковым статус-кодом.
- Реализована централизованная обработка ошибок в единой middleware. Все ошибки должны проходить через централизованный обработчик, отправка ошибок напрямую в контроллере запрещена.
- При обработке ошибок в блоке `catch` они не выбрасываются через `throw`, а передаются в централизованный обработчик ошибок с помощью `next`.