Чеклист для самопроверки №13

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

Работа отклоняется от ревью

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

В этом блоке собраны требования к работе, по которым вы можете выявить и самостоятельно исправить частые ошибки.

Файловая структура проекта:

- В проекте есть файл `package.json`:
  - в `package.json` корректно заполнены поля `name` и `author`;
  - `package.json` содержит все зависимости, используемые в проекте;
  - в `package.json` нет лишних зависимостей;
  - в `package.json` добавлены команды `start`, `dev` и команда `lint` с содержанием `npx eslint .`. При запуске команд отсутствуют ошибки.
- В проекте есть файл `.eslintrc`:
  - настройки `eslint` расширяют конфигурацию `airbnb-base`;
  - добавлено исключение для идентификатора `_id`.
- В проекте есть файл `.gitignore`, как минимум в этот файл добавлена папка `node_modules`.
- В проекте есть корректно заполненный файл `.editorconfig`;
- У проекта ясная структура, код разбит на роуты, модели и контроллеры, расположенные в соответствующих папках;
- Нет лишних файлов, которые не относятся к проекту.

Запуск проекта:

- Команда `npm run start` запускает сервер на `localhost:3000`;
- Команда `npm run dev` запускает сервер на `localhost:3000` с hot reload;
- При запуске команды `npm run lint` выполняется проверка проекта, в результате работы которой должны отсутствовать ошибки линтинга.

Работа приложения:

- При успешном ответе сервера возвращается `json`;
- При разных запросах сервер не падает и в консоли нет ошибок;
- При запуске приложение подключается к серверу mongo по адресу: `mongodb://localhost:27017/mestodb`;

- В приложении описана схема пользователя с полями:
  - `name` — строка от 2 до 30 символов, обязательное поле;
  - `about` — строка от 2 до 30 символов, обязательное поле;
  - `avatar` — строка, обязательное поле.
- В приложении описана схема карточки с полями:
  - `name` — строка от 2 до 30 символов, обязательное поле;
  - `link` — строка, обязательное поле;
  - `owner` — ObjectId, обязательное поле;
  - `likes` — массив ObjectId, по умолчанию пустой;
  - `createdAt` — дата создания, по умолчанию `Date.now`.
- Все поля схем пользователя и карточки валидируются;
- В файлах схем создаются и экспортируются модели с именами `user` и `card`;
- При обновлении пользователя или карточек в `options` передаётся `new: true`;
- Каждый запрос обрабатывается middleware, которая добавляет к объекту запроса поле `user`, содержащее объект с захардкоженным `_id` пользователя;
- Ответ сервера на запрос отправляется только один раз, например:

  Неправильно:

```
catch(err => {
  if (err.name === 'ValidationError') {
    next(err)
  }
  next(err)
})
```

  Правильно:

```
catch(err => {
  if (err.name === 'ValidationError') {
    next(err)
  } else {
    next(err)
  }
})
```

Приложение корректно обрабатывает запросы по следующим роутам:

- `GET /users` — возвращает всех пользователей из базы;
- `GET /users/:userId` — возвращает пользователя по `_id`;
- `POST /users` — создаёт пользователя с переданными в теле запроса `name`, `about` и `avatar`;
- `PATCH /users/me` — обновляет профиль пользователя;
- `PATCH /users/me/avatar` — обновляет аватар пользователя;
- `GET /cards` — возвращает все карточки из базы;
- `POST /cards` — создаёт карточку с переданными в теле запроса `name` и `link`, устанавливает поле `owner` для карточки;
- `DELETE /cards/:cardId` — удаляет карточку по `_id`;
- `PUT /cards/:cardId/likes` — ставит лайк карточке;
- `DELETE /cards/:cardId/likes` — убирает лайк с карточки.

Обработка ошибок в приложении:

- Если в любом из запросов что-то идёт не так, сервер возвращает ответ с ошибкой и соответствующим ей статусом:
  - **400** — переданы некорректные данные в методы создания карточки, пользователя, обновления аватара пользователя или профиля;
  - **404** — карточка или пользователь не найден или был запрошен несуществующий роут;
  - **500** — ошибка по умолчанию. Сопровождается сообщением: «На сервере произошла ошибка».
- Во всех контроллерах предусмотрена гарантированная отправка сообщения об ошибке;
- Статусы ошибок вынесены в константы;
- Ответ с ошибкой содержит только поле `message`. Сообщение об ошибке соответствует её типу;
- Нет обработки невозможных ошибок. Например, запрос на получение информации о пользователе не передаёт параметров, поэтому в нём невозможна ошибка 400. Такие кейсы предусматривать не нужно.