10.06.2020

+ Миграции

+ Seeds

Server

1. Проверяй подключение к базе сразу, как только приложение запустилось. Если база не работает - пусть приложение сразу падает и сообщает об ошибке. Используй метод db.authenticate() в index.js
2. Чтобы твой код можно было развернуть на любой машине - требуется синхронизировать таблицы и модели. Для этого требуется писать миграции:

<https://sequelize.org/master/manual/migrations.html>

**sync** метод хорош только на быструю руку, чтобы дропать базу и заново создавать. Но на реальных проектах он почти никогда не используется. Поэтому требуется описать миграции каждой из моделей. И новые поля и таблицы добавлять через новые миграции.

1. Чтобы каждый раз не заполнять базу с нуля, используй seeds. Опиши там моки (юзеров, книг и так далее), чтобы мне не пришлось с нуля базу заполнять.
2. Чем вызвано разделение юзера на Profile и User?

Profile.js

Когда у тебя одна таблица зависит от другой - требуется описывать ассоциации. Что бы user\_id точно ссылался на таблицу users. Иначе это просто цифра, которая почти ничего не значит.

  user\_id: {

    type: Sequelize.BIGINT, // no link to user!

    allowNull: false,

  },

References ???

User.js

Не нужно самому описывать поле с датой в моделях. Используй timestamps. <https://sequelize.org/v5/manual/models-definition.html>

updatedAt и createdAt описываются только в миграциях и сидах.

  date: { // can be replaced with createdAt, updatedAt

    type: Sequelize.DATE,

  },

users.js и auth.js используют одинаковый код для генерации jwt. Выноси генерацию в утилиты.

Выносить код из routes/api c (req, res) => … в папку /controllers

Унифицируй ответы от апи. Мы чаще всего используем подход с таким выводом:

{ error: boolean, data?: any, message?: string }

Frontend

1. Когда я зарегистрировал нового юзера и перешел в profile - у меня произошел запрос на /api/profile и вернулась 400 c profile not found. Я подозреваю что при создании юзера у тебя не создается запись профиля. Аналогично с незалогиненным юзером и запросом GET http://localhost:3000/api/auth 401 (Unauthorized)
2. Перемести компонент App из  /components выше, чтобы он лежал недалеко от index.js. В /components лучше хранить только те компоненты, которые переиспользуются. App более глобальная штука и среди футеров ему делать нечего.
3. Разделяй redux-thunk и код который отвечает за запрос (тот что связан с axios). Создай папку /api, настрой в одном из файле axios и используй его в дополнительных файлах. Например создать /api/auth.js с методами auth(), который возвращает промис.

Самим redux-thunk не нужно знать куда они делают запрос. Они просто выполняют какую-то функцию, которую в случае чего можно будет легко заменить на другую.

1. Не требуется каждый раз писать "Content-Type": "application/json",. Оберни axios через axios.create и задай ему значения по-умолчанию. Там же можешь подсовывать токен, если он есть через интерсепторы.

При регистрации юзера создавать профайл

Button/index.jsx

Используй props.children вместо props.value. Использование <Button>Hello world</Button> очевиднее и проще читается чем <Button value=”Hello world” />

Так же мне не понятно как ты будешь прокидывать прочие пропсы эту кнопку. Что если мне захочется кинуть onClick? А если потом еще что-то дополнительное? Тут явно нужен rest оператор.

Container/index.jsx и прочие вроде Form/index.js

Чем вызвана нужда завернуть контейнер в функцию? Можно ведь и без него обойтись.

const Container = ({ children }) => {

  return <StyledSection>{children}</StyledSection>;

};

const StyledSection = styled.section`

  margin: 0 auto;

  max-width: 300px;

`;

export default Container;

login-page.jsx + profile-page.jsx

Странице незачем знать про стейт формы. Саму форму можно вынести в отдельный компонент, прописать там всю логику валидации и просто пробрасывать метод onSubmit. Пусть форма как-то работает сама по себе и просто иногда выполняет метод onSubmit взятый из пропсов, когда все провалидировано верно.

Тащить валидацию в страницу плохо, потому что если там будет много всего - файл получится очень длинный.

 Нет смысла делать анонимную функцию заново, можно просто прокидывать onSubmit

<Form onSubmit={(e) => onSubmit(e)}> // onSubmit

reducers/auth.jsx

С localstorage лучше работай не в редьюсерах, а в redux-thunks или в api функциях. Редьюсеры должны быть максимально чистые и только данные перегонять из одного в другое. Без сайд эффектов.

reducers/books.jsx

Если тебе придется добавить новое поле в стор - твой редьюсер начнет сыпать багами. Не забывай копировать прошлые поля и перезаписывать те которые нужно. Проверь это также в других файлах на случай если я где-то пропустил.

    case ACTION\_TYPES.booksRequested:

      return {

        books: [],

        loading: true,

        error: null,

      };