Integração



Instalar dependências necessárias

npm install

Executar os projetos

node .\index.js – localhost:5000

npm start – localhost:3000

Contexto do backend

endpoints

/api/login – realizar ‘login’ e gerar um token de acesso

{

"email" : "admin@email.com",

"password" : "123456"

}

/api/postits

GET – listar postits

POST – cadastrar postit

PUT – atualizar postit

DELETE – deletar postit

Contexto do frontend

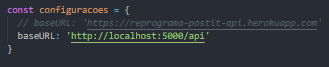
Mostrar todas as chamadas e como elas são feitas utilizando a API já pronta.

**actions.js**

baseURL : <https://reprograma-postit-api.herokuapp.com>

Alterar URL para API local

baseURL : <http://localhost:5000/api>

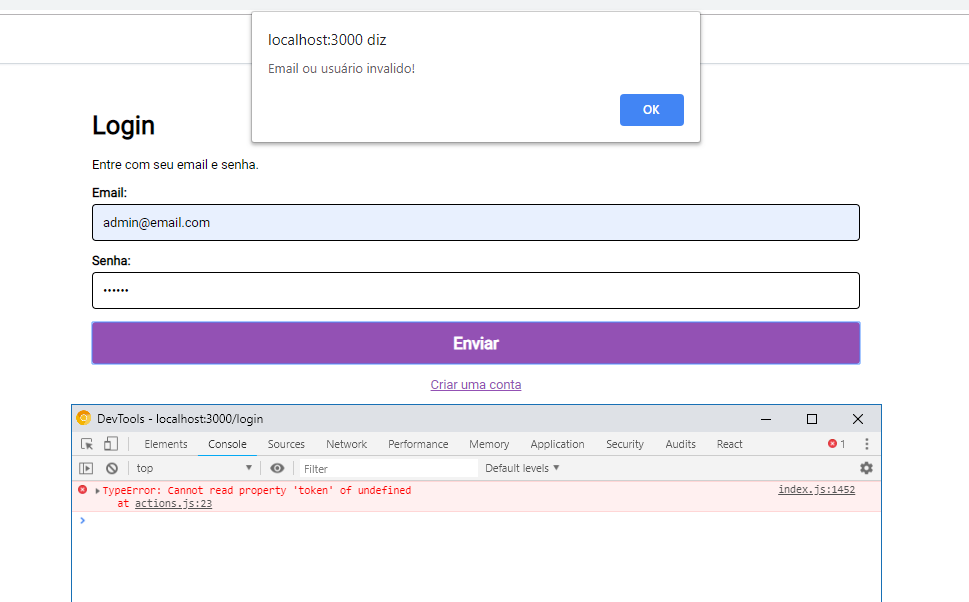


Irá ocorrer um erro de usuário e senha inválidos. Isso ocorre pq o backend recebe as informações de email/password e o que você está enviando, são as informações de email/senha.

**actions.js**



Mesmo encaminhando as informações corretas, ainda assim, ele continua mostrando as informações de email ou usuários inválidos. Por quê isso ocorre?



Isso ocorre por que a maneira como o token está sendo guardado, não está correta.

Atualmente ele recebe a informação de response.data.usuario.token. Vamos alterar para que no retorno, ele guarde diretamente a informação do token.

.then(response => {

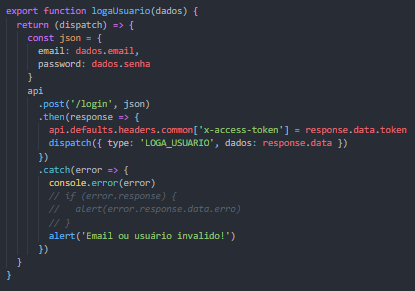
api.defaults.headers.common['Authorization'] = response.data.token

dispatch({ type: 'LOGA\_USUARIO', dados: response.data })

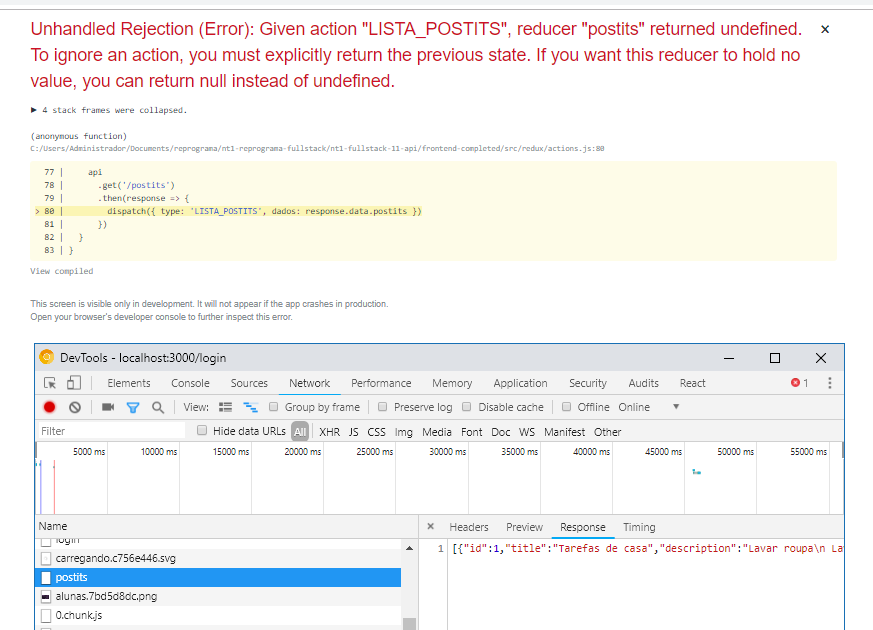
})

Além disso, vamos já perceber que o back-end não trabalha com o campo no cabeçalho de Authorization, mas sim, com o campo de ‘x-access-token’.

No mesmo arquivo **actions.js,** vamos alterar para que tanto a informação passada quanto guardada, seja do *x-access-token*.



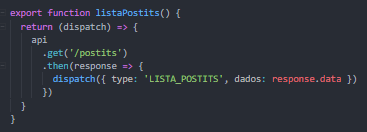
Vamos perceber que mesmo o retorno tendo sido 200, um erro ocorreu no front.



Isso ocorre por que inicialmente o frontend esperava uma chave com o nome postits, porém, ele não recebe essa atualmente, trabalhando somente com a resposta.

Vamos alterar para que a lista de postits chame somente o response.data.

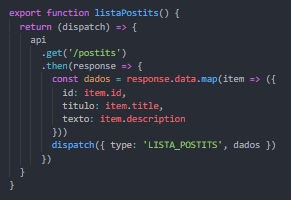
**actions.js**



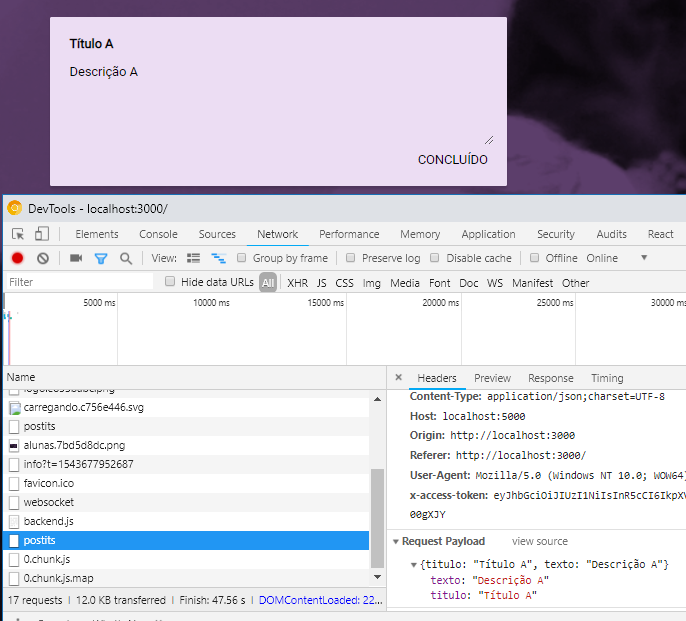
Ainda sim, as informações do back-end, não são mostradas na tela. Por quê?

Repare nas chaves de retorno da antiga API e repare nas chaves da atual API.

**actions.js**

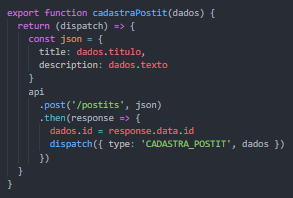


Cadastrar um novo postit

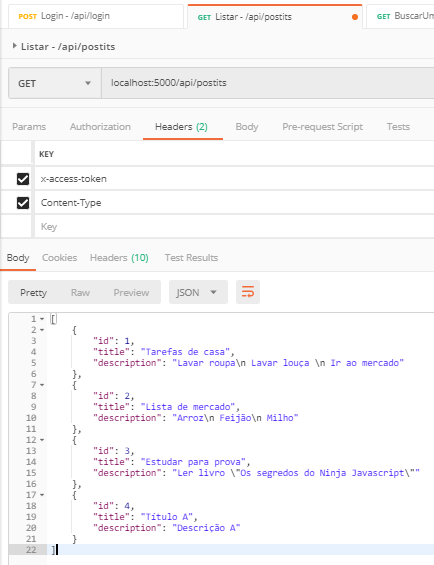


Ocorreu um erro devido a novamente estarmos recebendo title/description no backend, porém, as chaves que estamos enviando competem a texto/título.

**actions.js**

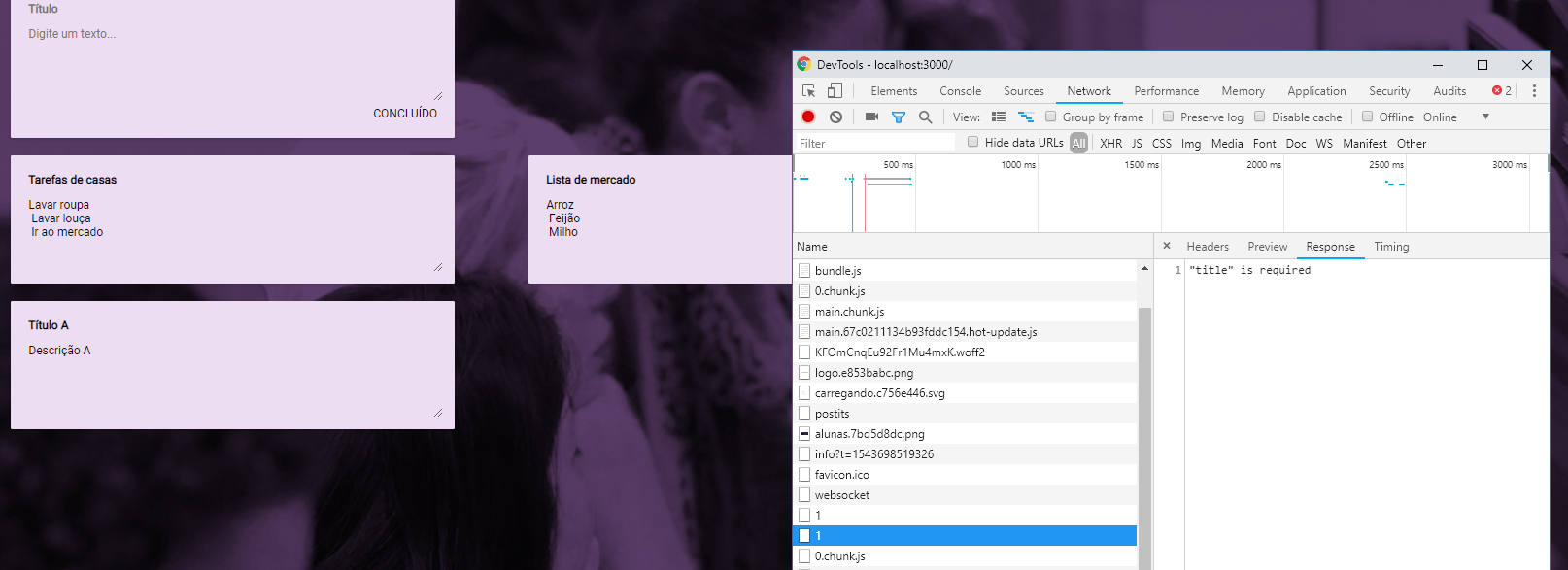


Listando os dados do backend.

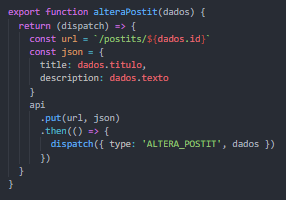


Atualizando um postit

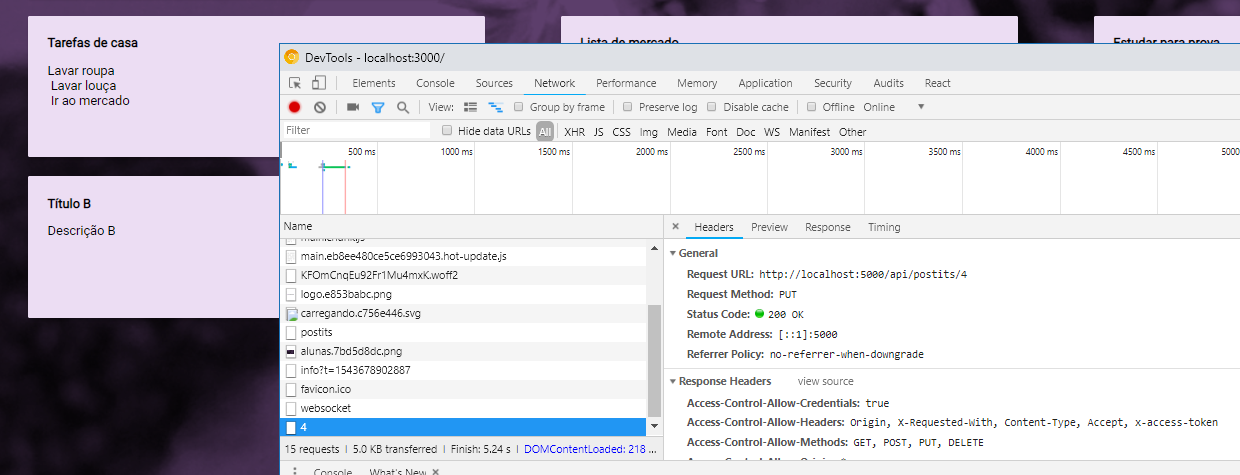
Ao tentarmos atualizar um determinado postit:



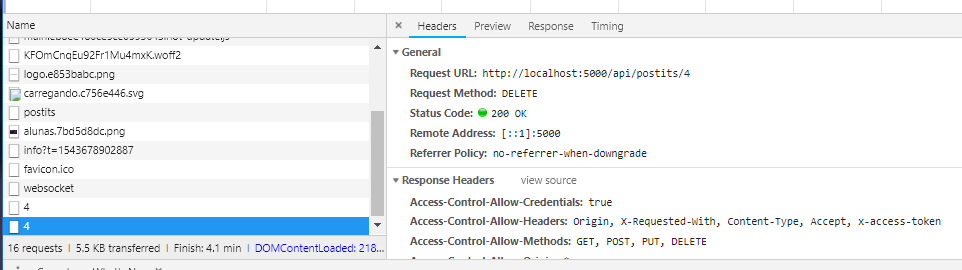
Vamos montar o corpo da requisição:



Sendo assim, ao realizarmos uma nova atualização, tudo irá ocorrer com sucesso.



Deletando um postit.



Contexto do back-end + contexto do mongodb

Por enquanto, nossa lista está em uma lista local fixa no back-end. Segurança, confiabilidade, perda dos dados.

Vamos alterar o back-end para que a partir de agora, ele armazene todas as nossas informações no back-end. Para esta comunicação, vamos utilizar o mongoose.

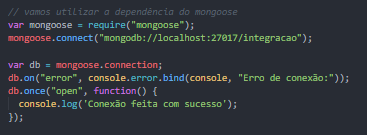
Mongoose

npm install --save mongoose

Contexto conexão com o banco

**index.js**

Realizando a conexão com o banco



Contexto do postit

**postits/postits.js**

Altero o meu postits para que a lista não fique mais local. Mas que ela trabalhe com os dados do banco de dados.

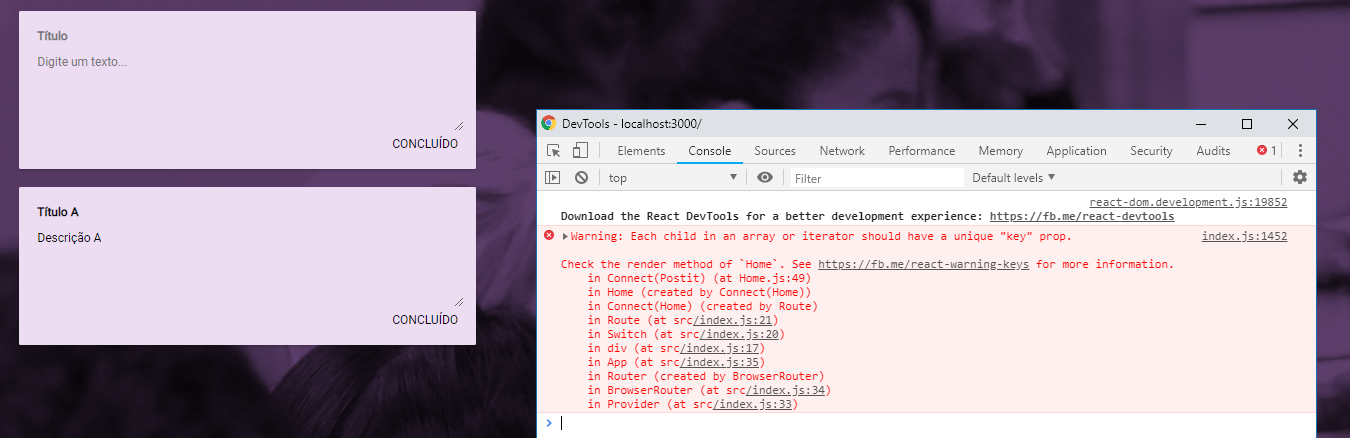


Inserir alguns postits direto no banco.

**postits/routes.js**



Vamos testar se o front continua funcionando.



Por que a chave não está sendo carregada?

Estamos recebendo da API, agora o \_id, e carregando o id. Precisamos também mapear para que ela receba o \_id.



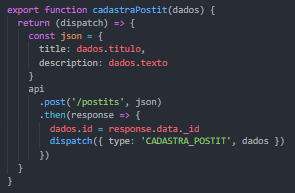
Vamos alterar para que ele salve as informações também no banco.

**postits/routes.js**



Vamos inserir pelo front, e verificar que precisamos ajustar um dos retornos.

**actions.js**



Buscando um postit, buscando pelo id.

**postits/routes.js**



Atualizando o postit.

**postits/routes.js**



Deletando um postit.

**postits/routes.js**



Contexto do usuário

**users/users.js**



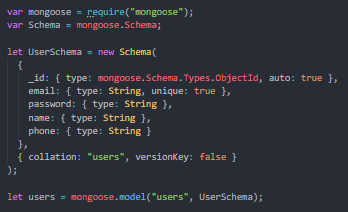
**index.js**



Integrando o cadastro

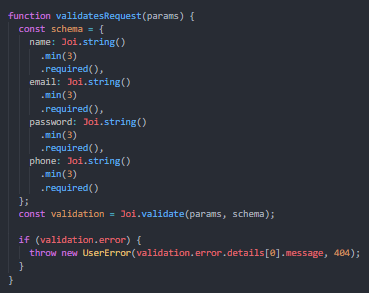
Back-end

**users/users.js**



**users/routes.js**





**frontend**

**Conta.js**



