BACKEND

Classes

Classe User:
Classe que define o usuário comum do website. É implementado como um ‘schema’ do mongoose.

Possui os seguintes atributos:

- **name**: string → Input obrigatório no cadastro
- **email**: string → Input obrigatório no cadastro | único
- **password**: string → Input obrigatório no cadastro | mínimo 6 caracteres
- **bookings**: [ ] → Lista de bookings (reservas) feitas

Classe Admin:
Classe que define o usuário administrador do website. É implementado como um ‘schema’ do mongoose.

Possui os seguintes atributos:

- **email**: string → Input obrigatório no cadastro | único
- **password**: string → Input obrigatório no cadastro | mínimo 6 caracteres
- **addedMovies**: [ ] → Lista de filmes adicionados no site

Classe Movie:
Classe que define os filmes adicionados no website. É implementado como um ‘schema’ do mongoose.

Possui os seguintes atributos:

- **title**: string → Input obrigatório no cadastro. Título
- **description**: string → Input obrigatório no cadastro. Descrição
- **actors**: [ ] → Lista de string’s de nomes. Atores participantes do filme
- **releaseDate**: date → Input obrigatório no cadastro. Data de lançamento
- **posterUrl**: string → Input obrigatório no cadastro. Url da imagem do pôster.
- **featured**: boolean → Se é destaque ou não
- **bookings**: [ ] → Lista de bookings feitos com este filme
- **admin**: admin → Parâmetro obrigatório. Admin que adicionou o filme no site

Classe Booking:
Classe que define as reservas que usuários fazem com um filme no website. É implementado como um ‘schema’ do mongoose.

Possui os seguintes atributos:

- **movie**: movie → Parâmetro obrigatório. Filme no qual a reserva será feita
- **date**: date → Input obrigatório no cadastro. Data da reserva

- **seatNumber**: number→ Input obrigatório no cadastro. Assento da cadeira reservada
- **user**: user → Parâmetro obrigatório. Usuário que fez a reserva

Controladores

Controlador User:

- **getAllUsers**( ) → Retorna todos os usuários comuns cadastrados, em formato .json
- **signup**(name, email, password) → Realiza o cadastro do novo usuário usando dos valores dos parâmetros e criptografando a senha, e retorna seu id em formato .json
- **updateUser**(id, name, email, password) → Busca o usuário que possua a mesma id recebida nos parâmetros e atualiza o nome, email e senha. Retorna uma mensagem em .json
- **deleteUser**(id) → Recebe o id do usuário a ser excluído e remove do sistema. Retorna uma mensagem em .json
- **login**(email, senha) → Encontra um usuário que possua o mesmo email que o fornecido, compara as senhas e retorna uma mensagem e o id do usuário logado, em formato .json
- **getUserById**(id) → Recebe o id do usuário a ser procurado e retorna o objeto user encontrado, em formato .json
- **getBookingsOfUser**(id) → Encontra o usuário com o id fornecido, e retorna todos os objetos bookings pertencentes a esse usuário, em formato .json

Controlador Admin:

- **addAdmin**(email, password) → Realiza o cadastro de um novo usuário admin com a senha criptografada, e retorna o objeto admin em formato .json
- **adminLogin**(email, password) → Encontra um admin que possua o mesmo email que o fornecido, e compara as senhas. cria um token jwt com duração de 1 dia e retorna em formato .json uma mensagem, o token e o id do admin logado
- **getAdmins**( ) → Retorna todos os admins cadastrados em formato .json
- **getAdminById**(id) → Recebe o id do admin a ser procurado, e retorna o objeto admin encontrado, em formato .json

Controlador Movie:

- **addMovie**(token, title, description, releaseDate, posterUrl, featured, actors) → Verifica se o token do admin está válido. Se sim, cadastra um novo filme com os inputs fornecidos e adicionando o filme no perfil do admin cadastrante. Retorna o objeto movie em formato .json
- **getAllMovies**( ) → Procura todos os filmes cadastrados e os retorna em formato .json
- **getMovieById**(id) → Recebe o id do filme a ser procurado, e retorna o objeto movie encontrado, em formato .json

- **deleteMovie**(token, id) → Verifica se o token do admin está válido. Se sim, usa o id fornecido para procurar e deletar o filme que possua o mesmo id. Retorna uma mensagem em .json

Controlador Booking:

- **newBooking**(movie, date, seatNumber, user) → Procura o filme e o usuário no banco de dados com os inputs. Realiza o cadastro do objeto booking e inicia uma sessão do mongoose para realizar as inserções de informações nos diferentes objetos. Retorna o objeto booking em formato .json
- **getBookingById**(id) → Recebe o id do booking a ser procurado, e retorna o objeto encontrado, em formato .json
- **deleteMovie**(id) → Encontra o booking a ser deletado com o id fornecido e inicia uma sessão do mongoose para remover o booking dos objetos movie e user em que participava. Retorna uma mensagem em .json

Routes

User Router

Os métodos controladores do objeto User são acessíveis através do link: "www.website.com**/user**", sendo seus sub links:

- **userRouter.get**( ) → .com**/user/** → Requisição GET retorna o método getAllUsers( )
- **userRouter.post**( ) → .com**/user/signup** → Requisição POST retorna o método signup( )
- **userRouter.put**( ) → .com**/user/:id** → Requisição PUT retorna o método updateUser( )
- **userRouter.delete**( ) → .com**/user/:id** → Requisição DELETE retorna o método deleteUser( )
- **userRouter.post**( ) → .com**/user/login** → Requisição POST retorna o método login( )
- **userRouter.get**( ) → .com**/user/:id** → Requisição GET retorna o método getuserById( )
- **userRouter.get**( ) → .com**/user/bookings:id** → Requisição GET retorna o método getBookingsOfUser( )

Admin Router

Os métodos controladores do objeto Admin são acessíveis através do link: "www.website.com**/admin**", sendo seus sub links:

- **adminRouter.get**( ) → .com**/admin/** → Requisição GET retorna o método getAdmins( )
- **adminRouter.post**( ) → .com**/admin/signup** → Requisição POST retorna o método addAdmin( )

- **adminRouter.post**( ) → .com**/admin/login** → Requisição POST retorna o método login( )
- **adminRouter.get**( ) → .com**/admin/:id** → Requisição GET retorna o método getAdminById( )

Movie Router
Os métodos controladores do objeto Movie são acessíveis através do link: "www.website.com**/movie**", sendo seus sub links:

- **movieRouter.get**( ) → .com**/movie/** → Requisição GET retorna o método getAllMovies( )
- **movieRouter.get**( ) → .com**/movie/:id** → Requisição GET retorna o método getMovieById( )
- **movieRouter.post**( ) → .com**/movie/** → Requisição POST retorna o método addMovie( )
- **movieRouter.delete**( ) → .com**/movie/:id** → Requisição DELETE retorna o método deleteMovie( )

Booking Router
Os métodos controladores do objeto Booking são acessíveis através do link: "www.website.com**/booking**", sendo seus sub links:

- **bookingRouter.post**( ) → .com**/booking/** → Requisição POST retorna o método newBooking( )
- **bookingRouter.get**( ) → .com**/booking/:id** → Requisição GET retorna o método getBookingById( )
- **bookingRouter.delete**( ) → .com**/booking/:id** → Requisição DELETE retorna o método deleteBooking( )

Explicação geral do App.js

1. O código começa importando os módulos necessários, incluindo o dotenv para configuração das variáveis de ambiente, o express para criar o servidor, o cors para lidar com a política de mesma origem e o mongoose para a conexão com o banco de dados MongoDB.

2. A configuração do dotenv é realizada para carregar as variáveis de ambiente.

3. O aplicativo Express é inicializado.

4. É configurado o CORS usando as opções definidas na variável corsOptions.

5. É configurado o body parser JSON para permitir o uso de JSON no corpo das requisições.

6. São importados os roteadores (userRouter, adminRouter, movieRouter e bookingsRouter) que serão responsáveis por lidar com as rotas relacionadas aos usuários, administradores, filmes e reservas, respectivamente.
7. As rotas são definidas para os roteadores importados, usando os caminhos base /user, /admin, /movie e /booking.

8. É estabelecida a conexão com o banco de dados MongoDB usando a URL de conexão fornecida no trecho de código.

9. Se a conexão com o banco de dados for bem-sucedida, o servidor é iniciado na porta 5000 e exibe a mensagem "Conectado ao servidor e servidor está funcionando".

10. Se ocorrer algum erro durante a conexão com o banco de dados, o erro é exibido no console.

FRONTEND

Api-helpers

- **getAllMovies**( ) → Se a requisição for bem-sucedida, retorna um array contendo os filmes cadastrados.
- **sendUserAuthRequest**(data, name, email, password, signup) → Se a requisição for bem-sucedida, autentica e cadastra usuários no backend e retorna os dados da resposta.
- **sendAdminAuthRequest**(data, email, password) → Se a requisição for bem-sucedida, autentica o admin e retorna os dados da resposta.
- **getMoviesDetails**(id) → Se a requisição for bem-sucedida, retorna os dados da resposta contendo os detalhes do filme.
- **newBooking**(data, movie, seatNumber, date, user) → Se a requisição for bem-sucedida, faz o agendamento do filme e retorna os dados da resposta.
- **getUserBooking**( ) → Se a requisição for bem-sucedida, retorna os dados da resposta contendo as reservas do usuário.
- **deleteBooking**(id) → Se a requisição for bem-sucedida, deleta a reserva do filme e retorna os dados da resposta.
- **getUserDetails**( ) → Se a requisição for bem-sucedida, retorna os dados da resposta contendo os detalhes do usuário.
- **addMovie** (data, title, description, releaseDate, posterUrl, featured, actors, admin) → Se a requisição for bem-sucedida, adiciona um filme com os parâmetros fornecidos e retorna os dados da resposta.
- **getAdminById**( ) → Se a requisição for bem-sucedida, retorna os dados da resposta contendo os detalhes do administrador.

- **deleteMovie**(id) → Se a requisição for bem-sucedida, deleta o filme com o id fornecido e retorna os dados da resposta.

Components

- **AlertError**( ) → Renderiza o elemento com o texto "AlertError".
- **AlertSuccessful**( ) → Renderiza o elemento com o texto "AlertSuccessful".
- **Admin**( ) → Componente responsável pela autenticação e login do administrador.
- **Auth**( ) → Componente responsável pela autenticação e login do usuário.
- **AuthForm**(onSubmit, isAdmin) → Renderiza diversos elementos que mudam com base nas propriedades ‘isAdmin’ e ‘isSignup’. Componente responsável por exibir o formulário de autenticação (login/signup) para usuários.
- **Booking**( ) → Renderiza detalhes do filme e seus detalhes, e do formulário de agendamento, além do botão de enviar. Componente responsável por exibir os detalhes de um filme e permitir que o usuário agende ingressos para o filme.
- **CradLayout**( ) → Componente responsável por exibir um card com informações de um filme, incluindo título, descrição, data de lançamento e imagem do pôster. Também fornece um botão para redirecionar o usuário para a página de agendamento do filme.
- **AddMovie**( ) → Componente responsável por exibir um formulário para adicionar um novo filme. O formulário inclui campos para inserir o título, descrição, URL do pôster, data de lançamento, atores e a opção "Featured" (destacado). Ao enviar o formulário, os dados são enviados para a API para adicionar um novo filme.
- **MovieItem**( ) → Componente responsável por exibir um item de filme em um layout de grade. Ele renderiza um cartão com uma imagem do filme, título e data de lançamento. Também possui um botão "Agendar" que redireciona para a página de reserva do filme.
- **Movies**( ) →Componente responsável por exibir uma lista de filmes. Ele busca todos os filmes usando a função getAllMovies da API e renderiza cada filme usando o componente MovieItem.
- **AdminProfile**( ) → Componente responsável por exibir o perfil do administrador. Ele busca as informações do administrador usando a função getAdminById da API e renderiza os detalhes do perfil, incluindo o nome de usuário e a lista de filmes adicionados pelo administrador. O administrador pode excluir os filmes da lista.
- **UserProfile**( ) → Componente responsável por exibir o perfil do usuário. Ele busca as informações do usuário usando a função getUserDetails da API e as reservas do usuário usando a função getUserBooking da API. O componente renderiza os detalhes do perfil do usuário, incluindo nome, email e uma lista de suas reservas. O usuário também pode excluir as reservas.
- **Header**( ) → Componente responsável por exibir o cabeçalho da aplicação. Ele contém a barra de navegação, pesquisa de filmes, ícone de menu (para dispositivos móveis) e um menu suspenso para autenticação e navegação.
- **HomePage**( ) → Componente responsável por renderizar a página inicial da aplicação. Ele exibe uma imagem de destaque, os últimos lançamentos de filmes e um botão para ver todos os filmes.

Store

- **userSlice**( ) → Este componente gerencia a sessão do usuário comum, permitindo o login e logout. O estado inicial indica se o usuário está logado ou não. O Redux Toolkit é utilizado para facilitar o gerenciamento do estado.
- **adminSlice**( ) → Este componente gerencia a sessão do usuário administrador, permitindo o login e logout. O estado inicial indica se o administrador está logado ou não. O Redux Toolkit é utilizado para facilitar o gerenciamento do estado.
- **store**( ) → é utilizada para criar a store do Redux. A store combina os reducers do userSlice e adminSlice para gerenciar o estado da aplicação. Essa configuração permite o acesso aos estados através da store para toda a aplicação.

Explicações gerais

Em **index.js**:
O ReactDOM é utilizado para renderizar a aplicação React no elemento HTML com o id "root". O axios é configurado com a URL base para as requisições HTTP. A aplicação é envolvida com os componentes <BrowserRouter> do React Router e <Provider> do Redux, que fornecem a funcionalidade de roteamento e gerenciamento de estado respectivamente. O componente <App /> é o componente raiz da aplicação.

Em **App.js**:
É o componente principal da aplicação App. Ele define as rotas da aplicação utilizando o react-router-dom e renderiza os componentes correspondentes a cada rota. Além disso, faz uso do Redux para gerenciar o estado de autenticação do usuário/administrador. Através do Redux, verifica se o usuário ou administrador estão autenticados e exibe as rotas correspondentes. Também realiza a verificação no localStorage para definir o estado de autenticação inicial ao carregar a aplicação. O componente também inclui o cabeçalho Header em todas as páginas.