BASIC FLOW

State of auth:

```
const initialState = {
  token: localStorage.getItem('token'),
  isAuthenticated: null,
  loading: true,
  user: null
}
```

Navbar renders (sign up, login) buttons when the isAuthenticated state is false (or null), (logout) button when the isAuthenticated state is true.

```
return (
  <nav className="navbar bg-dark">
    <h1>
      <Link to="/">
        <img className="main-logo" src={mainLogo} alt="dev connector main logo" />
      </Link>
    </h1>
    {!props.auth.loading && (
      <React.Fragment>
        {props.auth.isAuthenticated ? authLinks : guestLinks}
      </React.Fragment>
    )}
  </nav>
)
;
```

```
const authLinks = (
  <ul>
    <li>
      <Link onClick={props.logout} to="#!">
        <i className="fas fa-sign-out-alt"></i>{' '}
        <span className="hide-sm">Logout</span>
      </Link>
    </li>

  </ul>
);
```

```
const guestLinks = (
  <ul>
    <li>
      <Link to="#!">Developers</Link>
    </li>
    <li>
      <Link to="/register">Sign Up</Link>
    </li>
    <li>
      <Link to="/login">Login</Link>
    </li>
  </ul>
);
```

Auth reducer:

```
case REGISTER_SUCCESS:
case LOGIN_SUCCESS:
  return {
    ...state,
    ...action.payload,
    isAuthenticated: true,
    loading: false
  }
```

```
case REGISTER_FAIL:
case LOGIN_FAIL:
case AUTH_ERROR:
  return {
    ...state,
    token: null,
    isAuthenticated: false,
    loading: false
  }
```

```
case LOGOUT:
  return {
    ...state,
    token: null,
    isAuthenticated: false,
    loading: false
  }
```

```
  case USER_LOADED:
    return {
      ...state,
      isAuthenticated: true,
      loading: false,
      user: action.payload
    };
  default:
    return state;
}
```

SIGN UP

1. React renders REGISTER COMPONENT.

2. Sign up button fires handleSubmit(e).

3. In handleSubmit, invoke props.register({name, email, password}) if password === confirmPassword. register() is an action creator.

```
const handleSubmit = async event => {
  event.preventDefault();
  if (password !== confirmPwd) {
    props.setAlert('Passwords do not match!', 'danger');
  } else {
    props.register({ name, email, password });
  }
};
```

4. In register(), sends POST request to /api/users

```
export const register = ({ name, email, password }) => async dispatch => {
  const config = {
    headers: {
      'Content-Type': 'application/json'
    }
  }

  const body = JSON.stringify({ name, email, password });

  try {
    const response = await axios.post('/api/users', body, config);
```

5. In RegisterController() at POST api/users route, creates new user from the data user entered, and then invoke jwt.sign()

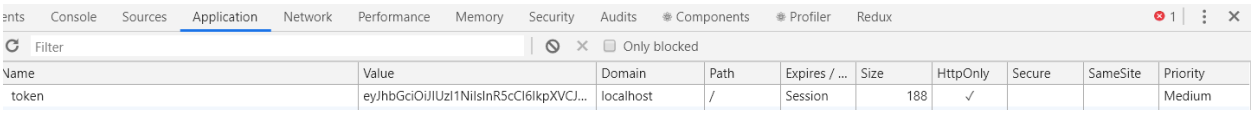
6. jwt.sign() signs a token and then sets cookie named 'token' and its value as the token jwt.sign() has created.

```
jwt.sign(payload, config.get('jwtSecret'), { expiresIn: '1h' }, (err, token) => {
  if (err) throw err;
  res.cookie('token', token, { httpOnly: true }).sendStatus(200);
});
```

7. After the token has successfully set, register() dispatches REGISTER_SUCCESS and loadUser().

```
dispatch({
  type: REGISTER_SUCCESS,
  payload: response.data
});
dispatch(loadUser());
```

8. In loadUser(), sends GET request to /api/auth with the token to be authorized.




```
export const loadUser = () => async dispatch => {
  try {
    const response = await axios.get('/api/auth');
```

9. In GET /api/auth route, server checks the token in the auth middleware, and confirms auth when the token is valid.

```
// @ROUTE          GET api/auth
// @DESCRIPTION    check authentication
// @ACCESS         Private
router.get('/', auth, CheckAuthController);
```

```
function auth(req, res, next) {
  const token = req.cookies.token;

  if (!token) {
    return res.status(401).json({ msg: 'No token, authorization denied' });
  }

  try {
    const decoded = jwt.verify(token, config.get('jwtSecret'));

    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({ msg: 'Token is invalid.' });
  }
}
```

↑ auth middleware

To get the cookie from request like “req.cookies.token”, you have to set cookie-parser middleware in the express app like this ↓

```
const cookieParser = require('cookie-parser');
```

```
app.use(cookieParser());
```

```
// @ROUTE          GET api/auth
// @DESCRIPTION    check authentication
// @ACCESS         Private
function CheckAuthController(req, res, next) {
  res.status(200).send();
}
```

↑ send status code 200 when the token validation succeeded (i.e. successfully passed auth middleware)

10. When the token validation has successfully done, loadUser() dispatches USER_LOADED. By checking the returned status code from GET /api/auth. Otherwise, dispatches AUTH_ERROR.

```
  if (response.status === 200) {
    dispatch({
      type: USER_LOADED,
      payload: response.data
    });
  }
} catch (err) {
  dispatch({
    type: AUTH_ERROR
  });
}
```

LOGIN

1. React renders LOGIN COMPONENT.

2. Login button fires handleSubmit(e).

3. In handleSubmit, invoke props.login(email, password) login() is an action creator.

4. In login(), sends POST request to /api/auth with stringified JSON data

```
const body = JSON.stringify({ email, password });

try {
  const response = await axios.post('/api/auth', body, config);
```

5. In LoginController() at POST api/auth route, check email and password and if the checking is successfully done, set 'token' cookie like we did in SIGN UP 6.

6. According to the result of auth process, login() dispatches LOGIN_SUCCESS and loadUser(), or LOGIN_FAIL

```
  dispatch({
    type: LOGIN_SUCCESS,
    payload: response.data
  });

  dispatch(loadUser());
} catch (err) {
  const errors = err.response.data.errors;

  if (errors) {
    errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));
  }
  dispatch({ type: LOGIN_FAIL });
}
```