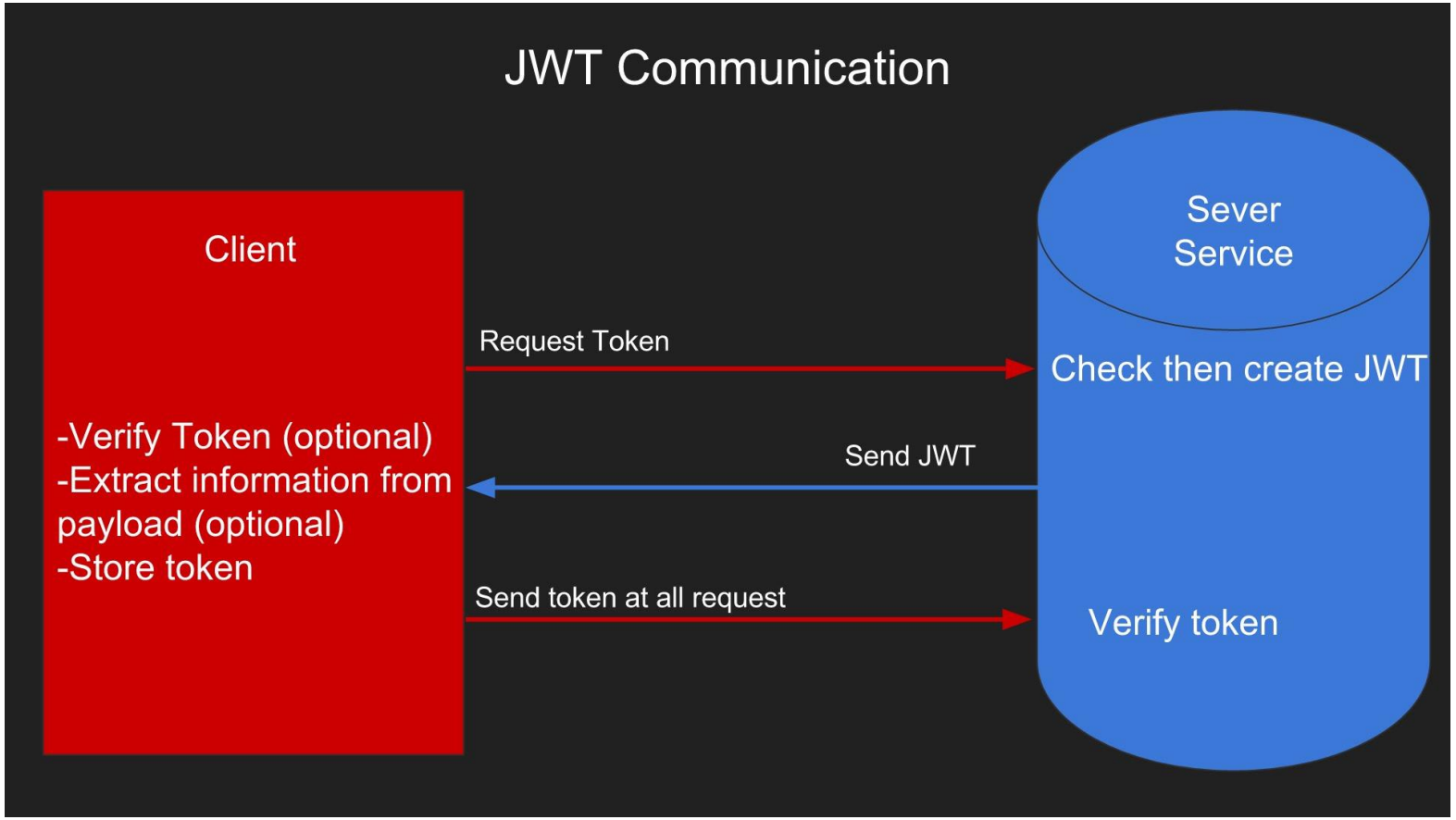
Authentication for Web APIs

using Javascript Web Tokens and Passport

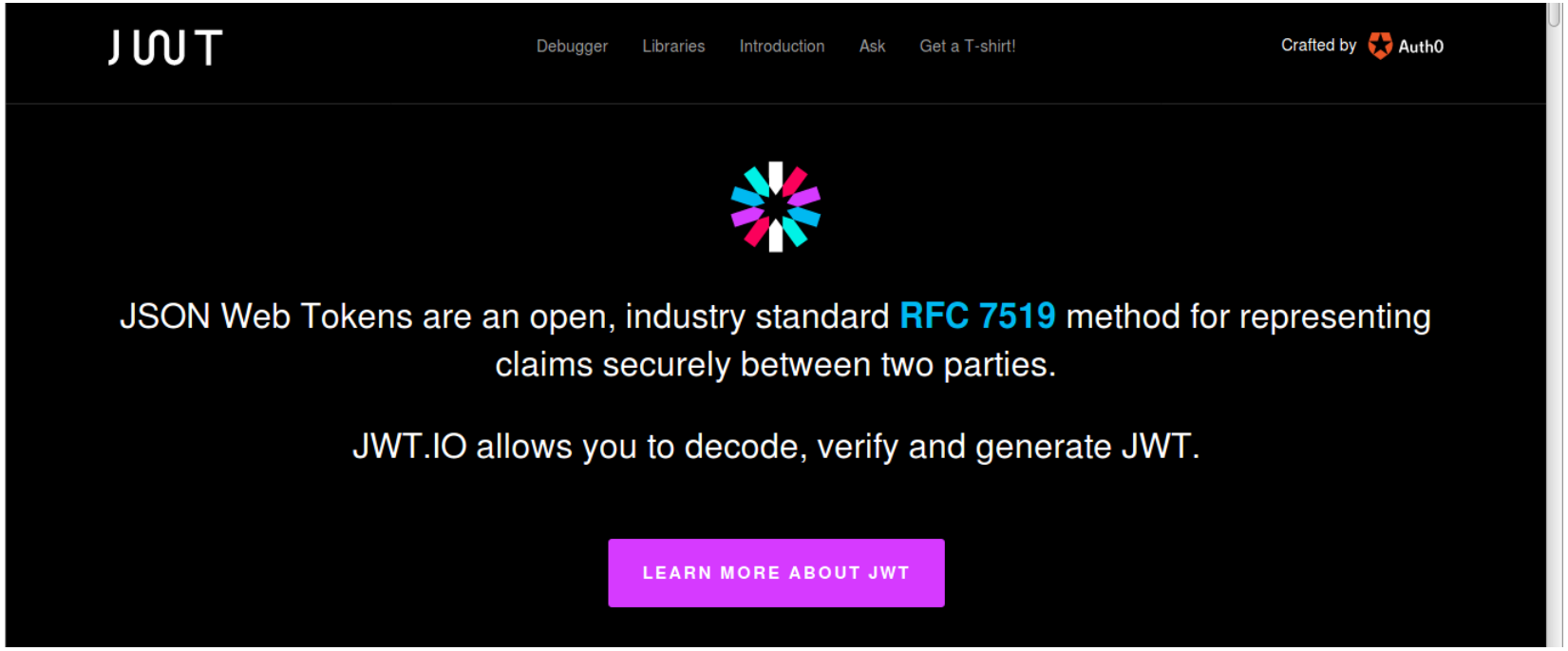
Frank Walsh, 2018

Authentication options

- Many solutions for Auth
  - Cookies, basic-auth, JWT
  - Web-based Identity Federation
- Jason Web Tokens (JWT)
  - Tokens means no need to keep sessions or cookies
  - In keeping with REST stateless principle – token sent on each request
  - Token stored on client, usually in local storage of client.



JWT Website


JWT
Debugger
Libraries
Introduction
Ask
Get a T-shirt!
Crafted by Auth0
JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.
JWT.IO allows you to decode, verify and generate JWT.
LEARN MORE ABOUT JWT

Username and Password Scenario

- Scenario
  - User signs up to access an API (username & password)
  - Create a new user in database
  - Use new username to create a JWT
  - Send JWT back to user
  - User stores JWT
  - JWT used on every subsequent request to protected resource
- Authentication and Identification
  - ...because username was used to generate JWT.

Authentication Middleware

- Need express middleware to manage user login
- Need Express middleware to restrict access to sensitive routes.
- Options
  - Roll our own
  - Use existing framework/package

```
app.use(function (req, res, next) {
    if (!userAuthenticated(req)) {
        return res.redirect('/login');
    }
    next();
});

app.use(express.static(__dirname + '/public'));
```

Passport

- Passport is authentication middleware
- Flexible and modular.
- Easy to retrospectively drop in to an Express app.
- Lots of "strategies" for authentication
  - Username/Password
  - Facebook
  - Twitter

Passport Overview

- Passport offers different authentication mechanisms as **Strategies**
  - You install just the modules you require for a particular strategy
- Authenticate by calling passport.authenticate()
  - specify which strategy to use.
- The authenticate() function signature is standard Express middleware

Authentication for "Hacker News"

- Restrict access to authenticated users.
- Provide user API to login/register.
- Users should only have to log in once:
  - Ideally identified and authenticated in subsequent requests.
- Username and Password authentication.
- No clear case passwords
  - Hash/Salt all passwords in MongDB

Create Mongoose User Model

Use Mongoose to specify user model:

```
1   import mongoose from 'mongoose';
2   import bcrypt from  'bcrypt-nodejs';
3
4   const Schema = mongoose.Schema;
5
6
7   const UserSchema = new Schema({
8     username: {
9           type: String,
10          unique: true,
11          required: true,
12      },
13    password: {
14          type: String,
15          required: true,
16      },
17  });
```

Mongoose Middleware: Hash/Salt Passwords

- Mongoose supports Middleware (also called pre and post *hooks*).
- Can use, like Express middleware, to process documents
- Use bcrypt package to hash and salt passwords

```
19  UserSchema.pre('save', function(next) {
20      const user = this;
21      if (this.isModified('password') || this.isNew) {
22          bcrypt.genSalt(10, (err, salt)=> {
23              if (err) {
24                  return next(err);
25              }
26              bcrypt.hash(user.password, salt, null, (err, hash)=> {
27                  if (err) {
28                      return next(err);
29                  }
30                  user.password = hash;
31                  next();
32              });
33          });
34      } else {
35          return next();
36      }
37  });
```

Mongoose Methods: compare passwords

- You can define instance and static methods in Mongoose Schemas.
- For authentication, define a comparePassword(..) instance method
  - Use this to authenticate users
  - Bcrypt used to compare with hashed/salted password.

```
39    UserSchema.methods.comparePassword = function(passw, cb) {
40        bcrypt.compare(passw, this.password, (err, isMatch) => {
41            if (err) {
42                return cb(err);
43            }
44            cb(null, isMatch);
45        });
46    };
47
```

User API: User Routes

- Create new router to support following API

Route	GET	POST	PUT	DELETE
/api/users	List all users	Register/ Authenticate User	N/A	N/A

User API: Get users

Create a route to list all users:

```
1   import express from 'express';
2   import User from './userModel';
3   import asyncHandler from 'express-async-handler';
4   import jwt from 'jsonwebtoken';
5
6   const router = express.Router(); // eslint-disable-line
7
8   // Get all contacts, using try/catch to handle errors
9   router.get('/', async (req, res) => {
10    try {
11      const contacts = await User.find();
12      res.status(200).json(contacts);
13    } catch (error) {
14      handleError(res, error.message);
15    }
16  });
```

User API: Register new user

- Will use query string of URL to indicate action to take on resource
  - action=register will register new user

```
19  router.post('/', asyncHandler(async (req, res) => {
20    if (!req.body.username || !req.body.password) {
21      res.json({
22        success: false,
23        msg: 'Please pass username and password.',
24      });
25    };
26    if (req.query.action === 'register') {
27      const newUser = new User({
28        username: req.body.username,
29        password: req.body.password,
30      });
31      // save the user
32      await newUser.save();
33      res.json({
34        success: true,
35        msg: 'Successful created new user.',
36      });
37    } else {
```

User API: Authenticate User

- Find user and compare password using user model
- Generate and return JWT token using username field
- Client to store JWT in local storage.

```
37      } else {
38        const user = await User.findOne({
39          username: req.body.username,
40        });
41        if (!user) return res.status(401).send({success: false, msg: 'Au
42        user.comparePassword(req.body.password, (err, isMatch) => {
43          if (isMatch && !err) {
44            // if user is found and password is right create a token
45            const token = jwt.sign(user.username, process.env.secret);
46            // return the information including token as JSON
47            res.json({
48              success: true,
49              token: 'JWT ' + token,
50            });
51          } else {
52            res.status(401).send({
53              success: false,
54              msg: 'Authentication failed. Wrong password.',
55            });
56          }
57        });
58      };
59    }));
```

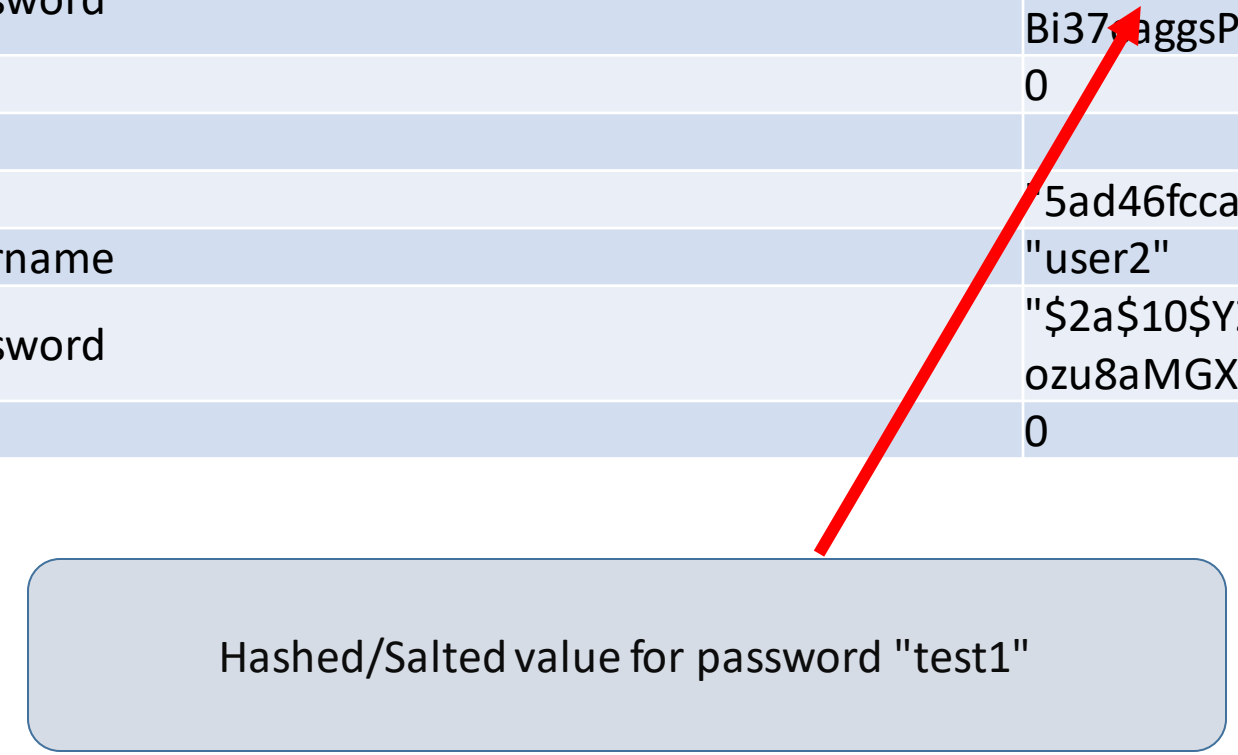
```
21      // #2 add a user
22      it('should register a user', function(done) {
23        // post to /api/contacts
24        supertest(app)
25        .post('/api/users')
26        .query({action: 'register'})
27        .send({username: 'Contact 99', password: 'test1'})
28        .expect('Content-type', /json/)
29        .expect(201)
30        .end(function(err, res) {
31          res.status.should.equal(201);
32          res.body.success.should.equal(true);
33          done();
34        });
35      });
```

```
37      // #3 login a user
38      it('should authenticate a user', function(done) {
39        // post to /api/contacts
40        supertest(app)
41        .post('/api/users')
42        .send({username: 'user1', password: 'test1'})
43        .expect('Content-type', /json/)
44        .expect(201)
45        .end(function(err, res) {
46          res.status.should.equal(200);
47          res.body.token.substring(0, 3).should.equal('JWT');
48          done();
49        });
50      });
51    });
```

User API: Testing, testing, testing...

Users API: User Collection

Users Collection	
_id	"5ad46fccada1ab2d67b349ec"
username	"user1"
password	"$2a$10$9r3v12AvPPSkcpJXiohGgehGY50gvgWFV9AAABi37aaggsPmxBdwW"
__v	0
1	
_id	"5ad46fccada1ab2d67b349ed"
username	"user2"
password	"$2a$10$YZlmbnUSZhBq9FAsAqKTyOJk8uXEweC7XtTNY/ozu8aMGXDW07Xxa"
__v	0



Protecting API Routes: Passport JWT Policy

- Passport strategies are a middleware functions that a requests runs through before getting to the actual route.
- If the authentication strategy fails,
  - callback will be called with an error
  - the route will not be called and a 401 Unauthorized response will be sent.

```
1   import passport from 'passport';
2   import passportJWT from 'passport-jwt';
3   import UserModel from './../api/users/userModel';
4   import dotenv from 'dotenv';
5
6   dotenv.config();
7
8   const JWTStrategy = passportJWT.Strategy;
9   const ExtractJWT = passportJWT.ExtractJwt;
10
11  let jwtOptions = {};
12  jwtOptions.jwtFromRequest = ExtractJWT.fromAuthHeaderAsBearerToken();
13  jwtOptions.secretOrKey = process.env.secret;
14  const strategy = new JWTStrategy(jwtOptions, async function(payload, next) {
15    console.log('payload received', payload);
16    // usually this would be a database call:
17    const user = await UserModel.find({username: payload});
18    if (user) {
19      next(null, user);
20    } else {
21      next(null, false);
22    }
23  });
24
25  passport.use(strategy);
26
27  export default passport;
```

Protecting API Routes: initialise and add Middleware

In root index.js of express

```
// import passport configured with JWT strategy
import passport from './auth';
…
// initialise passport
app.use(passport.initialize());
…
// Add passport.authenticate(..)  to middleware stack for protected routes
app.use('/api/posts', passport.authenticate('jwt', {session: false}),
  postsRouter);
```

```
5   describe('Hacker News Posts API unit test', function() {
6     this.timeout(120000); // eslint-disable-line
7     // #1 return a collection of json documents
8     it('should return collection of JSON documents', function(done) {
9       // calling home page api
10      supertest(app)
11      .get('/api/posts')
12      .set('Authorization', 'BEARER eyJhbGciOiJIUzI1NiJ9.dXNlcjE.FmYria8
13      .expect('Content-type', /json/)
14      .expect(200) // This is the HTTP response
15      .end(function(err, res) {
16          // HTTP status should be 200
17          res.status.should.equal(200);
18          done();
19      });
20    });
```

```
22    it('should deny access', function(done) {
23      supertest(app)
24      .get('/api/posts')
25      .expect('Content-type', /json/)
26      .expect(401) // This is the HTTP response
27      .end(function(err, res) {
28          // HTTP status should be 401
29          res.status.should.equal(401);
30          done();
31      });
32    });
33  });
34
```

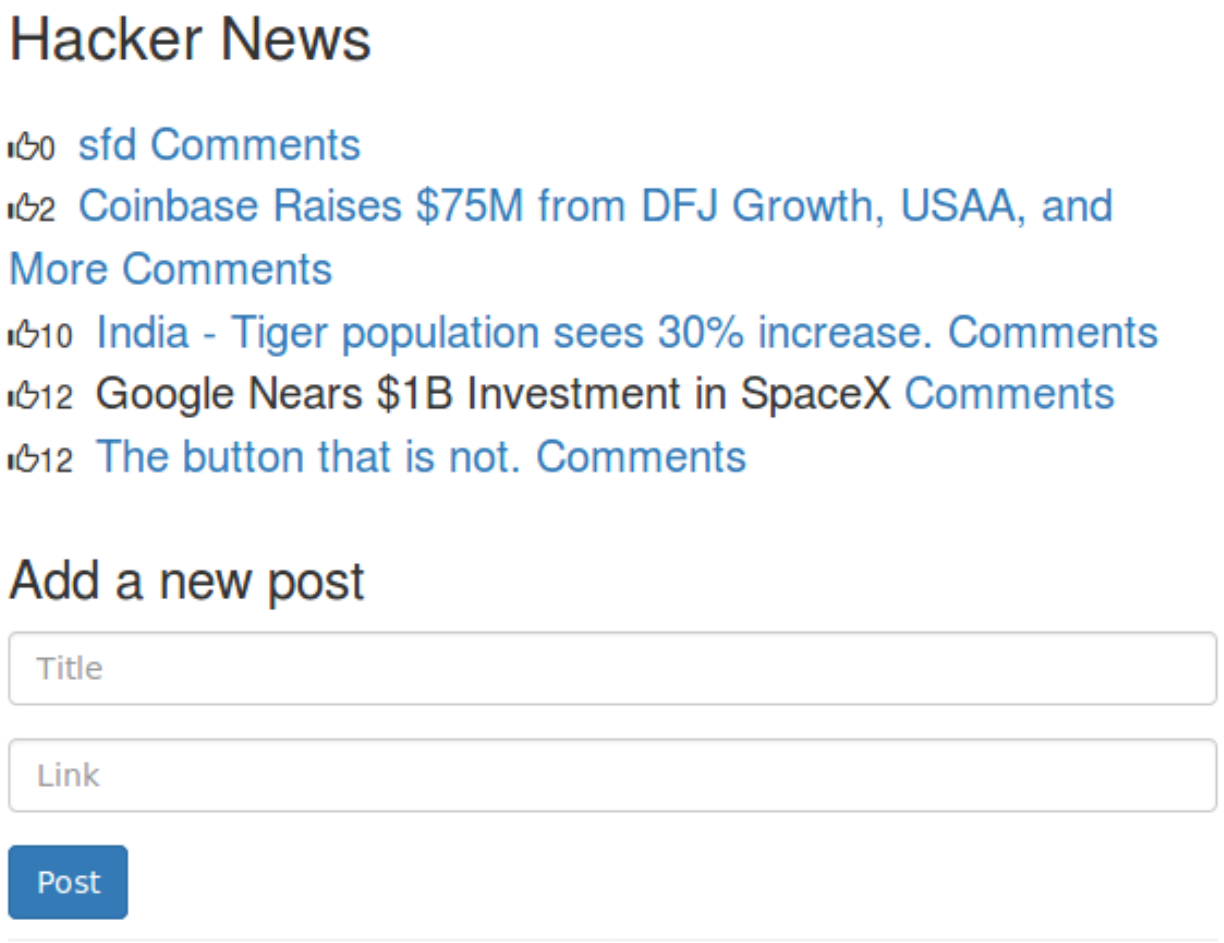
Protecting API routes: Testing

Requests for JWT protected routes must have a valid token in the HTTP **authorization** header.

Securing React App with JWT

Hacker News App

- Up to now used stubAPI or JSON server
- We want to:
  - Replace with calls to Express HackerNews API
  - Provide login/signin capabilities.
  - Only allow signed in users to see/add posts



Proposed Architecture

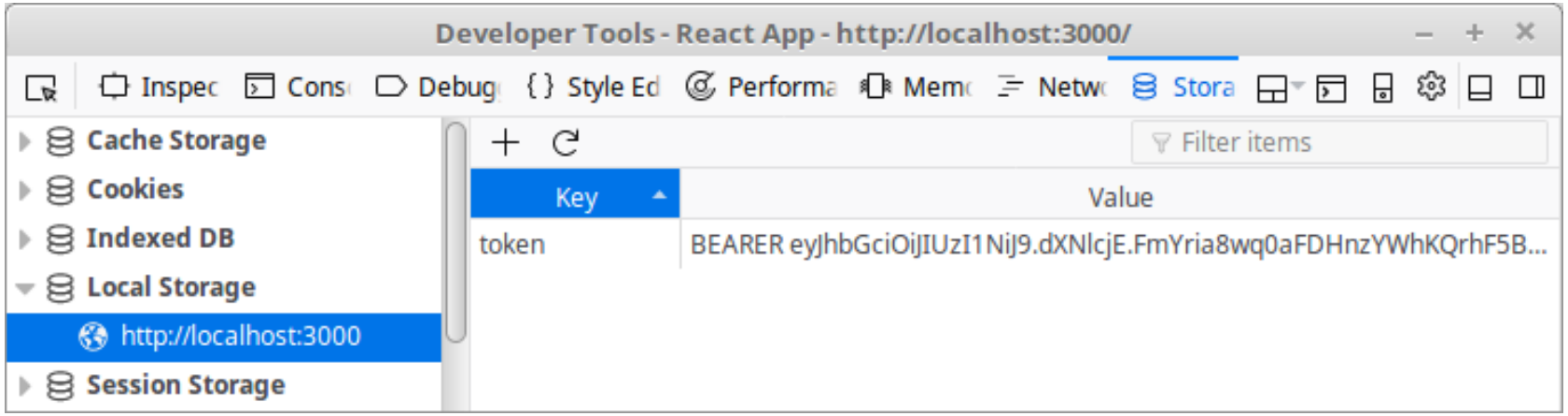
- Create-React-app uses Webpack development server.
- HackerNews API is an Express.js app.
- Configure Webpack server to "proxy" any unknown requests to Express app
  - Just need "**proxy":"http://localhost:8080"** entry in package.json.
- Removes Cross-Origin-Resource-Sharing (CORS) issues with the browser

JavaWebToken Storage

- Most browsers/devices have **local storage** .Can access using **localStorage** object.

```
localStorage.setItem('token', token);
```

```
const token =     localStorage.getItem('token');
```



Making API HTTP requests

- Use **Axios**, promise-based HTTP client to make requests to the Express API from the React app.
  - `npm install –save axios`

```
import axios from 'axios';
import auth from './auth';
export const upvote = async (postId) => {
    axios.post(`/api/posts/${postId}/upvote`)
                .then(resp => resp.data);
};

export const getAll = async () => {
    const resp = await axios.get('/api/posts',{headers: {'Authorization': auth.getToken()}},)
    return resp.data;
};

export const getPost = async (postId) => {
  const resp = await axios.get(`/api/posts/${postId}`,{headers: {'Authorization': auth.getToken()}})
  return resp.data;
};

export const add = async (newTitle, newLink) => {
  const resp = await axios.post('/api/posts',{title: newTitle, link: newLink }, {headers: {'Authorization': a
  return resp.data;
};
```

Integrating with React App

- Use HackerApp component state for posts.
- Use **componentDidMount()** function to the HackerApp component to initialise the posts from the Express API.
  - React Lifecycle Method that runs after all the elements rendered correctly

```
96   class HackerApp extends React.Component {
97    state = {posts: [{}]};
98    async componentDidMount () {
99      try{
100           const resp = await api.getAll();
101           this.setState({
102                    posts: resp,
103                    isHidden: false
104                  });
105
106         } catch (e){
107           this.setState({
108                    isHidden: true
109                  });
110         }
111   };
```

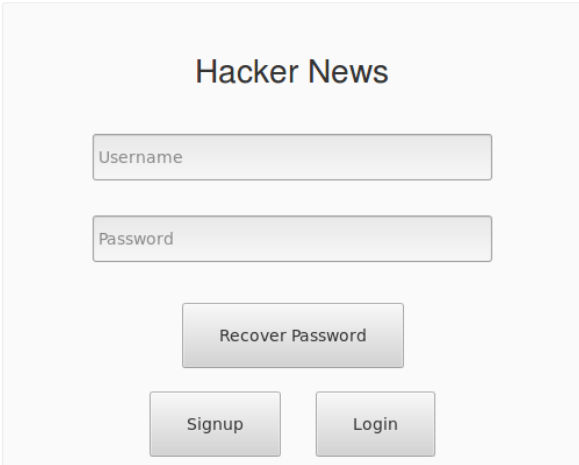
```
133    render() {
134      const posts = _.sortBy(this.state.posts, post =>
135           post.upvotes);
136      return (
137        <div >
138        {this.state.isHidden &&  <Button href="/login">Login/Signup</Butto
139        {!this.state.isHidden && <NewsList posts={posts}
140              upvoteHandler={this.incrementUpvote} />}
141          <Form addHandler={this.addPost} />
142       </div>
143      );
144  }
145  }
```

Login/Signin page

- No need to roll your own...
  - npm install --save react-signup-login-component
- Create **loginPage.js** and add to App router (in index.js)

```
18  const loginWasClickedCallback = async function(data) {
19      try{
20       const result = await api.login(data.username, data.password);
21       Auth.authenticateUser(result.token);
22       browserHistory.push('/');
23
24     }catch (e){
25         alert(`SAuthentication Failed: ${e}` )
26     }
27
28      };
29
30      return (
31          <div>
32              <ReactSignupLoginComponent
33                  title="Hacker News"
34                  handleSignup={signupWasClickedCallback}
35                  handleLogin={loginWasClickedCallback}
36              />
37          </div>
38      );
39  };
40
41  export default LoginPage;
```

```
26  import LoginView from './loginPage';
27
28  ReactDOM.render(
29   (
30    <Router history={browserHistory} >
31      <Route path="/" component={App}>
32         <IndexRoute component={HackerApp}/>
33         <Route path="posts/:postId" component={CommentView} />
34         <Route path="login" component={LoginView} />
35      </Route>
36    </Router>
37  ),
38    document.getElementById('root')
39  );
```



Summary

- Create User model with Mongoose
  - Pre-save hook to salt/hash passwords
  - Instance method to compare passwords
- Implement user API to authenticate/signup users
  - Sign JWT tokens with user name
- Add a JWT Strategy to Passport.js
- Use passport.authenticate(…) to secure server-side routes
  - Add to middleware stack.