Full Stack Development for Web Applications

James Rowley

Women Techmakers Tucson Hackathon 2017

Workshop Goals

- Cover most important concepts of data-based web apps
- Show how to develop one in its entirety (full-stack)
- NOT think too much about theory or absolute best practices
- Provide attendees with a framework they can successfully hack on and build their project from, even if half of the topics went over their heads

- I will not be covering in any great detail:
  - Front-end design work (HTML, CSS)
  - Server administration
  - Database systems

What is Full Stack?

- It depends who you ask…
- Generally refers to one developer or team handling front end and back end development together.
- May or may not include graphic design and/or system administration

The Learning Experience, Today

- This workshop is primarily structured around looking through a strategically unfinished web app, and filling in and talking about the blanks.

- If you haven't already, you should download the starter project from: https://goo.gl/x9Qj9U

The Project

General Information

Web App Goals

- “/”, Home Page – “Splash Screen” displaying a remotely updatable message retrieved from the server. Capability to navigate to the Control Page.
- “/actions”, Control Page – Demonstrate three concepts:
  - Multiple logical systems can operate independently on the same page, namely:
  - “Compute” – Sends a user-specified number and mathematical operation to the server and retrieves and displays the result, without navigating away or reloading.
  - “Set Splash” – Sends an arbitrary string to the server which will be used as the HTML for the splash screen going forward.
- Several other endpoints not aimed at end user

Our Stack

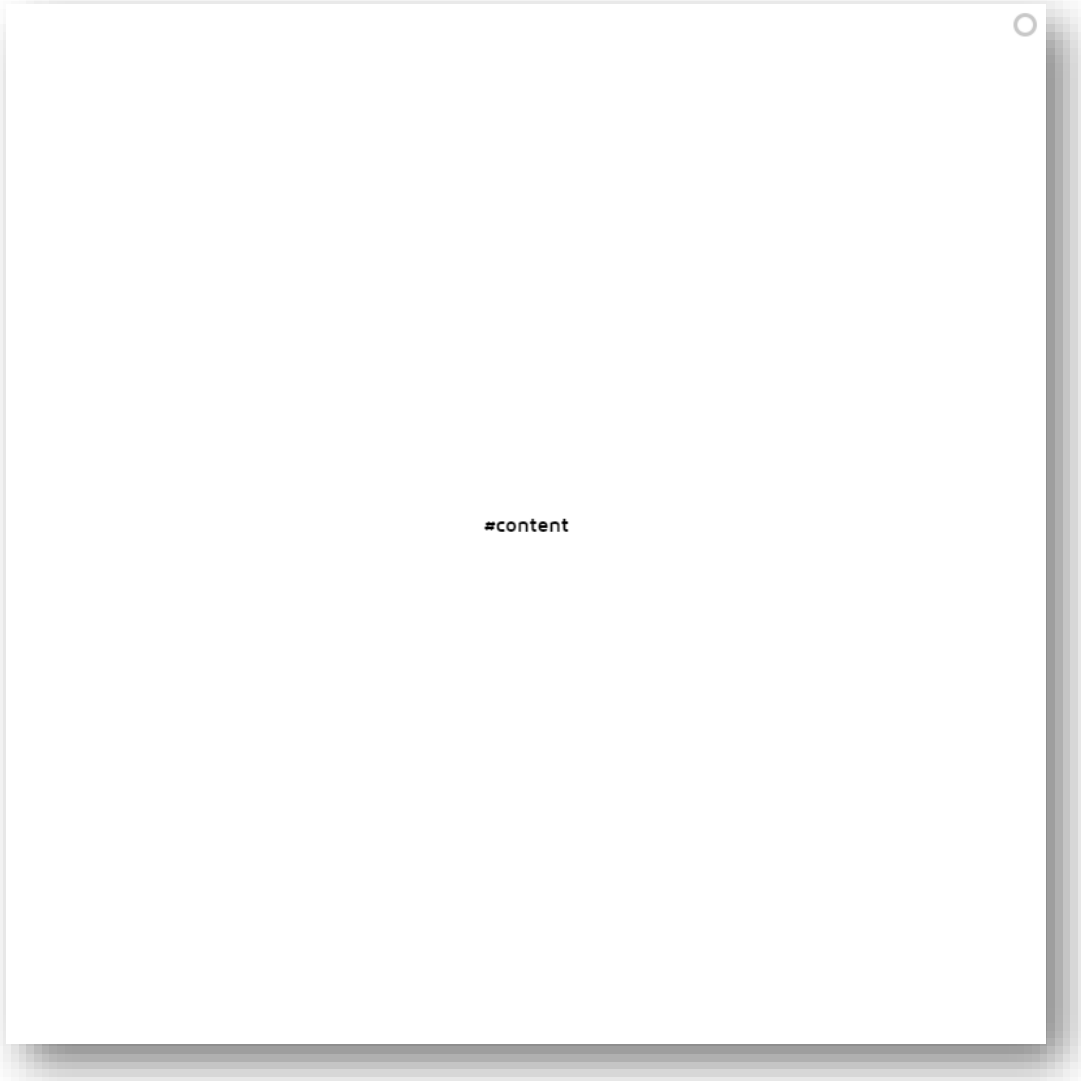
- Based on Node.js and Express
  - Node is nice because you only have to know JavaScript for frontend and backend
  - Express is popular, works well enough
- Using SQLite3 for database
  - Simple and quick
- Using Bootstrap for frontend layout and styling (mostly)
  - Simple and quick
- Using jQuery for frontend interaction
  - Simple and quick, makes verbose JS terse

Things to Note

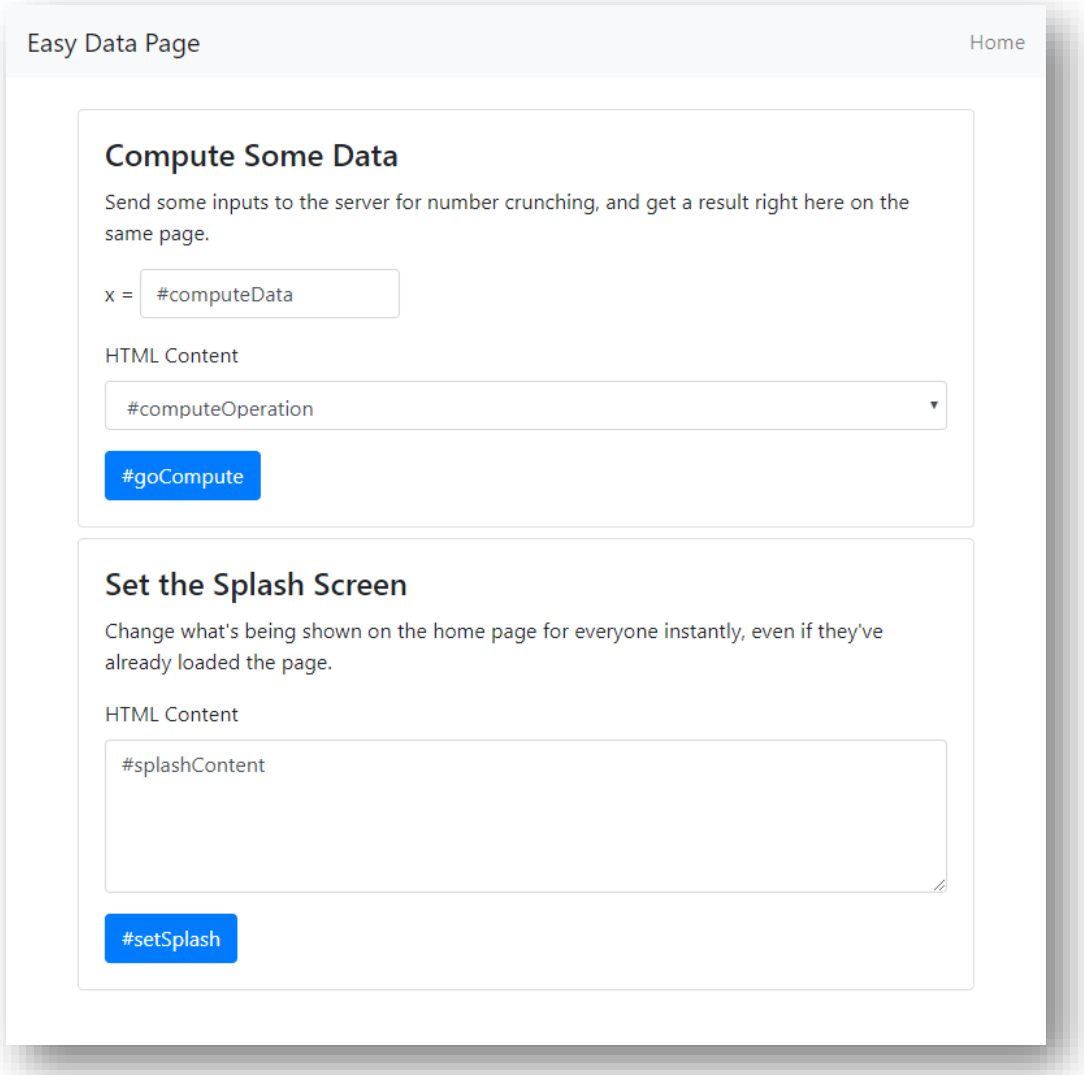
- Run the app by calling “node ./bin/www” from the base directory
- Webserver setup is in “www”
  - Amalgamation of examples and Stack Overflow posts, don’t look at it too hard
- Express setup is in app.js
  - Database is initialized here
  - Routes are registered here
  - Basically the base file

- There’s a file “COPYPASTING.txt” in the base directory with the code from these slides, to make your life a little easier

“/” – Splash: Important elements



“/actions” – Actions: Important elements



Filling In The Blanks

Application Setup

Set Up Database

/app.js

```
var bodyParser = require('body-parser');
var sqlite3 = require('sqlite3');
…
var db = new sqlite3.Database("splashes.db");
db.serialize(function(){
    db.run("CREATE TABLE IF NOT EXISTS splashes (id INTEGER,
        content TEXT, PRIMARY KEY(`id`));");
});
…
var app = express();
app.set('views', path.join(__dirname, 'views'));
```

Add Routes

/app.js

```
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
…
app.use('/', require('./routes/splash'));
app.use('/actions', require('./routes/actions'));
…
app.use(function(req, res, next) {
    var err = new Error('Not Found');
```

Filling In The Blanks

“Compute” Action

Data Model

- Client POSTs JSON to “/actions/compute” as:

{data: number, operation: string in [“fac”, “ex”, “sine”]}

- Server performs operation, returns JSON as:

{result: number}

- Later, if there was some error server side, it will return instead:

{error: string}

- NB: Client code will be similar to provided code for Set Splash

Client code to interact with server

/javascripts/actions.js

```
function computeData(data, operation, callback) {
    $.post(
        "/actions/compute",
        {data: parseInt(data), operation: operation},
        function(result) {
            if (typeof callback === 'function')
                callback(result.result);
        },
        "json"
    );
}
```

Server code to compute data

/routes/actions.js

```
router.post('/compute', function(req, res) {
    x = parseFloat(req.body.data);
    result = 0;
    switch (req.body.operation) {
        case "fac":
            result = factorial(x);
            break;
        case "ex":
            result = Math.pow(Math.E, x);
            break;
        case "sine":
            result = Math.sin(x);
            break;
    }
    res.send({result: result});
});
```

Client code to interact with page

/javascripts/actions.js

```
$("#goCompute").click(function(){
    computeData(
        $("#computeData").val(),
        $("#computeOperation").val(),
        function(result){
            $("#computeResult").text(result).show();
        }
    );
});
```

Server code to catch errors (/routes/actions.js)

```
router.post('/compute', function(req, res) {
    if (!(req.body.data && req.body.operation)) {
        res.status(400);
        res.send({error: 'malformed request'});
        return;
    }

    x = parseFloat(req.body.data);
    result = 0;
    switch (req.body.operation) {
        case "fac":
            result = factorial(x);
            break;
        case "ex":
            result = Math.pow(Math.E, x);
            break;
        case "sine":
            result = Math.sin(x);
            break;
        default:
            res.status(400);
            res.send({error: 'invalid operation'});
            return;
    }
    res.send({result: result});
});
```

Client code to handle errors

/javascripts/actions.js

```
function computeData(data, operation, callback) {
    $.post(
        "/actions/compute",
        {data: parseInt(data), operation: operation},
        function(result) {
            if (result.error) {
                console.log(result.error);
                return;
            }
            if (typeof callback === 'function')
                callback(result.result);
        },
        "json"
    );
}
```

Filling In The Blanks

"Splash Screen"

Data Model

- Setting:
  - Client POSTs JSON to “/actions/splash” as: {content: string}
  - Server performs operation, returns HTTP 200
- Receiving
  - Client requests HTTP GET “/splashcontent”
  - Server returns JSON with current splash content as: {content: string}

  -and/or-
  - Server preloads view “splash.ejs” with current splash content

- NB: Server code will be somewhat similar to provided code for Compute

Client code to set Splash Content

/javascripts/actions.js

- Already provided, nearly the same as that for the Compute feature

Server code to set Splash Content

/routes/actions.js

```
var db = new sqlite3.Database("splashes.db");
…
router.post('/splash', function(req, res) {
    if (typeof req.body.content !== 'string') {
        res.status(400);
        res.send({error: 'malformed request'});
        return;
    }

    var stmt = db.prepare(
        'INSERT INTO splashes (content) VALUES ((?));');
    stmt.run(req.body.content);
    stmt.finalize();

    res.status(200);
    res.send();
});
```

Client code to display Splash Content

/javascripts/splash.js

```
$(function(){
    setSplash(preload.content);
    window.setInterval(checkSplashContent, 5000);
});

function checkSplashContent() {
    $.get(
        "/splashcontent",
        setSplash,
        "html"
    );
}
```

Server code to send Splash Content

/routes/splash.js

```
function getSplashContent(id, after) {
    var query = 'SELECT content FROM splashes ORDER BY id DESC LIMIT 1;';
    if (typeof id === 'number') {
        query = 'SELECT content FROM splashes WHERE id=\''+id+'\' LIMIT 1;';
    } //Allow future possibility of selecting a specific splash

    db.all(query, function(err, content){
        if(!err)
            after(content[0]);
    });
}
…
router.get('/splashcontent', function(req, res, next) {
    getSplashContent(null, function(content){
        res.type('html');
        res.send(content.content);
    });
});
```

Success!

Hopefully!

Just run “node .\bin\www”.

Next Things to Try

- Easy:
  - Add more calculations and/or parameters to the “Compute” feature.
- Intermediate:
  - Add a way to select what splash is displayed, or maybe when it is.
- Hard:
  - Add authentication so only you and your closest friends can access “/actions”.
    - Recommend the Passport module and Google or Facebook authentication

- *Most importantly, build your cool web based project, and win!*

Thanks for coming!

Any questions?