Ten
Thousand
Coffees

Engineering Onboarding at Ten Thousand Coffees

Table of Contents

How We Work

Engineering at Ten Thousand Coffees (10KC) follows a very simple framework: we operate on sprints planned in a product sprint planning meeting, we execute retrospectives, and occasionally pair. Our product practice follows a lean approach, where user validation is critical, and the user is the central focus of our product decisions.

It is important to have a strong understanding of the core technologies we use at 10KC as well as practice strong engineering discipline. We believe in practicing clean code, owning the quality of our product, and practicing empathy to both our users and our co-workers.

- Lean Design practice: https://lean-product-design.18f.gov/index.html
- Clean Code for JavaScript: https://github.com/ryanmcdermott/clean-code-javascript
- Clean Code: http://ricardogeek.com/docs/clean_code.html
- How to give good feedback: https://medium.mikeindustries.com/how-to-give-helpful-product-design-feedback-1e4c053b6da
- Balanced Teams: https://www.youtube.com/watch?v=Z_Q4Q8rCVpU

The Basics

The core Ten Thousand Coffees (10KC) product is based on a **MEAN** stack: ***MongoDB***, ***Express.js***, ***Angular*** and **Node.js**. Currently, both versions of Angular are in use, however we are phasing out our Angular 1.5 application.

git

We use **git** for version control of our source code with our repositories hosted on **GitHub**. Our git workflow follows a feature-branch workflow, known as **GitHub Flow**. All of our commits on must be signed with a ***GPG*** signing key added to ***GitHub***.

- Introduction to git: https://www.atlassian.com/git/tutorials/what-is-git
- The GitHub Flow: https://guides.github.com/introduction/flow/
- Setting up commit signing: https://help.github.com/articles/signing-commits-with-gpg/

Asynchronous Operations

In ***TypeScript*** and ***JavaScript*** execution is singly-threaded, meaning every line executes after the other. In order to not halt the execution of your application, asynchronous operations must be used for long running calls such as network requests or database operations. Function callbacks are the simplest form of returning from asynchronous operation but should be avoided in favour of ***Promises, Observables***, or **Async**/**Await**. In Angular, ***Observables*** are preferred over ***Promises***.

- Understanding asynchronous JavaScript in 7 Seconds: https://twitter.com/manekinekko/status/855824609299636230?lang=en
- Understanding asynchronous JavaScript in 4 Minutes: https://medium.freecodecamp.org/understanding-asynchronous-javascript-callbacks-through-household-chores-e3de9a1dbd04
- Understanding promises: https://developers.google.com/web/fundamentals/getting-started/primers/promises
- Chaining Promises: https://spring.io/understanding/javascript-promises
- Understanding observables: https://dev.to/supermanitu/understanding-observables

API organization, CRUD and REST

APIs at 10KC are modeled to be ***REST***ful (Representational State Transfer) ***API***s. This means that ***CRUD*** (create, read, update, delete) operations have ***API*** endpoints that are consistent, stateless, and have an easily understood structure. ***API***s at 10KC must return ***JSON***, and endpoints must follow the jsonapi.org standard.

- Understanding JSON: https://spring.io/understanding/JSON
- Understanding CRUD: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
- Understanding RESTful APIS: https://spring.io/understanding/REST
- The JSON API source of truth: http://jsonapi.org/
- API Best Practices: http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

MongoDB

MongoDB is a ***NoSQL*** database where data is kept as ***documents*** held together in ***collections***. At 10KC, operations on the MongoDB database are abstracted in Node using the ***mongoose.js*** driver. ***Mongoose.js*** provides a ***schema*** based approach to modeling objects throughout our application. ***Documents*** can be embedded inside other ***documents***, this is known as ***denormalization***. MongoDB allows for CRUD operations, as well as complex, multi document operations known as ***aggregations*** that are executed in data pipeline.

- Understanding NoSQL: https://www.mongodb.com/nosql-explained
- Introduction to mongoose.js: http://mongoosejs.com/docs/guide.html
- Denormalization with MongoDB: https://docs.mongodb.com/manual/core/data-model-design/
- MongoDB Aggregations: https://docs.mongodb.com/manual/aggregation/

Express.js

Express is a framework for Node that gives you the ability to map an HTTP ***method***, with a ***path*** known as a ***route***, to a ***middleware*** function. Express conveniently packages and sends the HTTP request and parameters ***(req)*** to the ***middleware***, and gives an object to write the response to ***(res)***. ***Middlewares*** are often chained together, it is common to have the last ***middleware*** be a function belonging to a ***controller***. Chaining ***middlewares*** is helpful for separating functional areas like token validation, parameter validation, user account fetching, database connections, or error handling. It is possible to write an entire web application with express serving html content, however, it's use at 10KC is limited to serving the ***REST***ful ***API***, serving the static AngularJS ***webpack bundle,*** and ***server side rendering*** the Angular application.

- Understanding Express routing: http://expressjs.com/en/guide/routing.html
- Creating middlewares: http://expressjs.com/en/guide/writing-middleware.html
- Using middlewares: http://expressjs.com/en/guide/using-middleware.html
- Creating an Express CRUD Example: https://zellwk.com/blog/crud-express-mongodb/ and https://zellwk.com/blog/crud-express-and-mongodb-2/

Angular

Angular is a front-end web ***single page application (SPA)*** framework. The underlying architecture of Angular applications is **Model-View-Whatever (MVW)**. Angular provides elements known as ***components*** and ***directives***, that with a **template,** allow you to compose HTML elements with JavaScript, CSS and HTML. Data, known as ***properties***, can be set in the template through ***binding,*** which can either be ***one-way bound***, or **two-way *bound***. It is convention that objects known as ***services*** fetch data and provide it to **components**, these ***services*** are provided to the ***component*** through ***dependency injection***.

Routing between pages is handled within the application, by setting which ***component*** should resolve to the ***route*** provided to the ***router*** by the browser. Since ***routing*** is handled within the app, it is important to have the server ***mod_rewrite*** or similar to redirect routes back to the main page ***bundle***.

Angular has been completely re-written since it's inception, hence the distinction between Angular and AngularJS. Angular is the latest version, and AngularJS is the legacy version. At 10KC, our new application development is in Angular, however we still require infrequent maintenance of our legacy AngularJS codebase.

- Model-View-Whatever: https://plus.google.com/+AngularJS/posts/aZNVhj355G2
- Understanding modrewrite in Express: https://stackoverflow.com/questions/16579404/url-rewriting-with-expressjs

Angular (Angular 6+)

Angular is written in TypeScript, and needs to be compiled into a JavaScript version that is consumable by most web browsers. We use **@angular/cli** to do all the compilation for our projects to be browser ready. At 10KC, we ***server-side render*** our angular application meaning an html page is rendered on the server and served to our client, after the html loads the browser starts to load our main ***bundle*** and control is switched from the server to the ***bundle*** once it finishes loading.

- Work through the angular tour of heros: https://angular.io/tutorial
- Understand Angular Routing: https://angular.io/guide/router
- Angular CLI: https://cli.angular.io/
- Server-side rendering: https://angular.io/guide/universal
- Server-side rendering with Firebase: https://hackernoon.com/deploying-angular-universal-v6-with-firebase-c86381ddd445

Node.js

Node.js is an asynchronous JavaScript ***runtime*** that is backed by the ***Chrome V8 Engine***. Node treats HTTP as a first class citizen, and makes non-blocking IO calls. Node has the ability to add on ***frameworks*** and ***utilities*** as ***packages***, through the use of the ***require*** keyword in JavaScript and the ***npm*** package manager. There are two popular package managers that are used with node, ***npm*** and ***yarn***, however 10KC uses ***npm***. The command line utility **nvm** is useful for maintaining and updating your version of Node.

- Node Version Manager (nvm): https://github.com/creationix/nvm
- Understand the Node event loop: https://medium.com/the-node-js-collection/what-you-should-know-to-really-understand-the-node-js-event-loop-and-its-metrics-c4907b19da4c

Testing

Testing at 10KC is done through a combination of ***unit***, ***integration*** and ***manual*** tests. Quality is the responsibility of everyone on the team at 10KC. For the both the front end and the API, tests are written using the **Jest** framework, this covers ***assertions***, ***spying***, ***stubbing,*** and ***mocking***.

- Understanding TDD: Test-Driven Development
- Testing Using Jest: https://jestjs.io/docs/en/getting-started.html
- Testing Angular applications: https://angular.io/guide/testing

Lodash

Lodash is a very powerful JavaScript utility library that has been heavily optimized. If you need a utility for operating on an object, Lodash probably provides it. Lodash also allows for ***implicit chaining***, to optimize the output of multiple Lodash calls on the same objects. At 10KC, Lodash is preferred over its predecessor, UnderscoreJS.

- Lodash Documentation: https://lodash.com/docs
- Implicit Chaining: https://blog.mariusschulz.com/2015/05/14/implicit-function-chains-in-lodash

JSON Web Tokens

JSON Web Tokens (***JWT***s) are a way of transporting information using a JSON object that can be verified against a secret token. ***JWT***s consist of a ***header***, ***payload*** and ***signature***, the ***payload*** contains information known as ***claims***. The ***header*** and ***payload*** can be verified using the ***signature***, which is signed using a secret key upon creation. ***JWT***s are used at 10KC for authentication and user specific linking to events.

- Understand JWTS: https://jwt.io/introduction/

Putting it all together - The Bootcamp

Now you understand (at least a little) node.js, express.js, angular.js, MongoDB. All of these pieces work together to make up a stable, scalable, maintainable website development stack that we use at 10KC.

Your Challenge

Create a simple micro blog website using all the frameworks/tools you just learned. The microblog must:

- Support adding/deleting Entries to/from your Blog
- Support text up to 160 characters and an image per entry.
- Store any uploaded images in the mongodb.
- Support multiple Blogs that you are able to switch between.

The micro blog development must adhere to the following:

- The client side code should use Angular. (Note: when building the Angular app it is recommended that the **@angular/cli** be used to create the project)
- The server with use Node.js and Express.js
- The authentication should use stateless JWT's http://jwt.io/ https://github.com/auth0/angular2-jwt
- The database will be built with MongoDB and mongoose.js
- Use git for source control on a public repository

Submission

The source code for the microblog is to be provided to us at 10KC by making the repository available to us.