Architecture

Compare and contrast the types of frontend development you used in your full stack project, including Express HTML, JavaScript, and the single-page application (SPA).

Initially, our application architecture was an Express/Node.js application which used Handlebars templating engine to retrieve and display JSON files for quick prototyping. Next, I set up a MongoDB server and an Express/Node.js RESTful back-end within the encapsulating express application, able to interact with the database through HTTP calls and perform Create, Read, Update, Delete by creating request URL paths and parameters within our app\_api/routes/index.js file. We separate concerns into a MVC (Model, View, Controller) architecture, with routes forward incoming requests to their respective controllers, also written in Express logic, and enable dynamic rendering of the view via Handlebars templating. The controller can accept a parsed request and generate a response such as 200 for success or 404 for not found. We’re able to consume our API by calling it to pull lists of data and render it on the page. In our case, the architecture with the API and the application logic are both wrapped in the same Express logic, which makes it easy to make calls between the Express/NodeJS RESTful API and our front-end Express/NodeJs/Handlebars express app. Thus I had a structure consisting of app\_api/controllers, app\_api/models, app\_api/routes and app\_server/controllers, app\_server/routes, and app\_server/views. Finally I recreated the Node.js/Express/Handlebars front-end as a Angular SPA, interacting with the REST API to create and read trip listings using DRY principals. Node.js having an event-driven non-blocking IO model makes it very fast when programming asynchronously.

The whole Angular application resides in the app\_admin directory. By using Angular command line to create components, I can effectively modularize the front-end application into several interworking subcomponents such as trip-card, trip-listing, and add-trip which can all interact with a trip-data service to make HTTP requests to the API (which can reach our MongoDB database instance) and return a promise.

Why did the backend use a NoSQL MongoDB database?

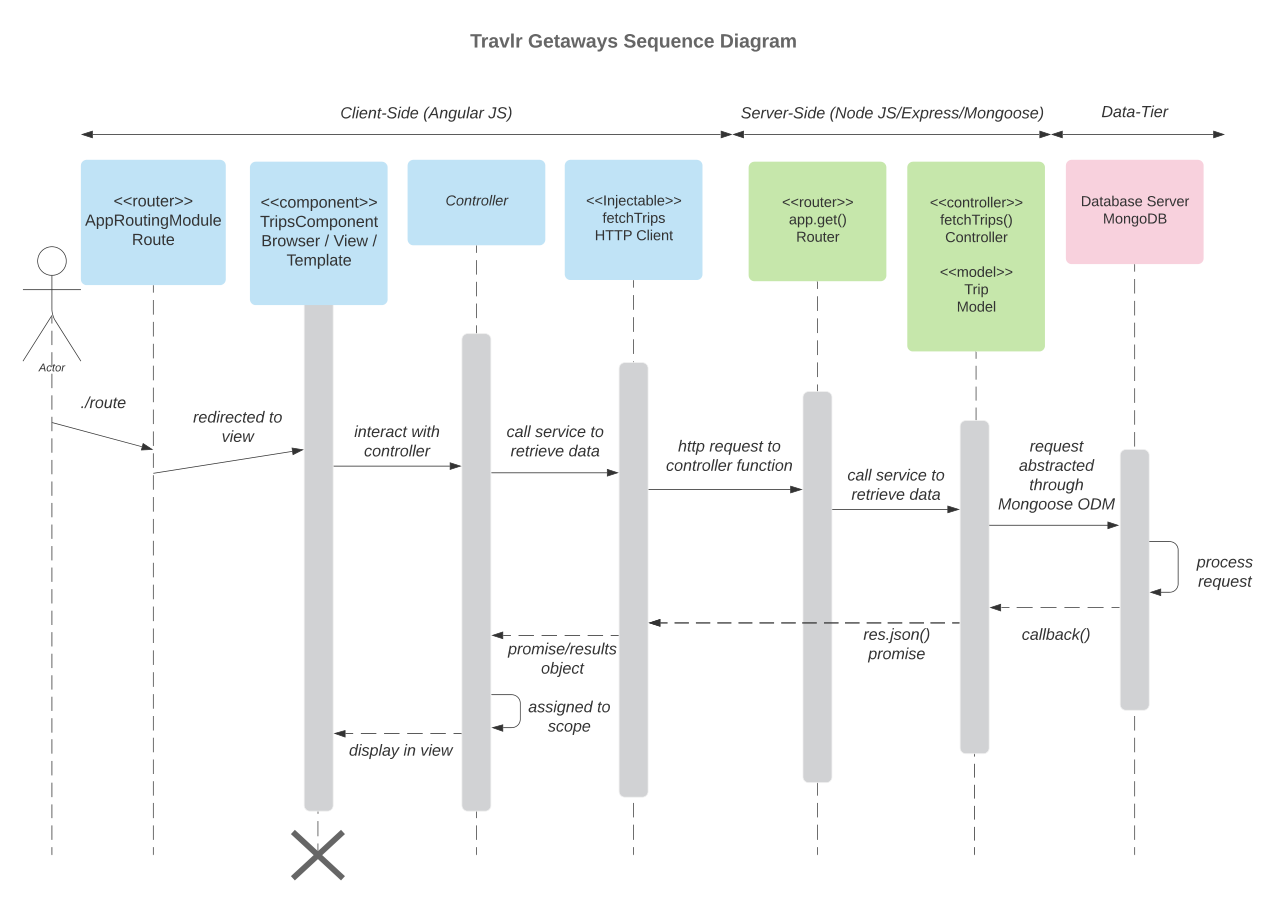
As NoSQL database is horizontally scalable, scalability is cheaper than relational databases and better performance can be achieved by simply adding a shard to the server, rather than buying better hardware. MongoDb uses a nested JSON structure for its collections so that we can compartmentalize similar data in a few collections, and offers great performance for mass reads and writes (except when updating a lot of collections regularly). It’s schema-less nature gives it great flexibility, and using mongoose we can define our document structure and properties using Mongoose schemas. For an app with many reads and few writes, performace is also high. MongoDB stores the data in BSON format which can then be exposed as JSON through Mongoose, fulfilling the “javascript everywhere” paradigm. In addition, if Geocapabilities are necessary Mongodb offers convenient geolocation functions to bolster our API. Note that for Data which is strongly related, stored in multiple collections, and updated often — SQL is best.

Functionality

How is JSON different from Javascript and how does JSON tie together the frontend and backend development pieces?

Javascript objects values can be any datatype, including functions, which is not possible with JSON. JSON supports Strings, Numbers, Objects, Arrays, Booleans, and null value. Additionally, all key names and strings must be double quoted.

We can use a Scenario to show how JSON ties the front-end and backend together.



A user entering the Travlr website URL to their browser. After entering their credentials Javascript runs and hits the router of the front-end server, which will call the TripsComponent and redirect its view to the browser. TripsComponent will interact with the controller, loading fetchTrips as a dependency injection service to retrieve Trips data. fetchTrips will create an HTTP request to the back-end server which reaches the RESTful endpoint router on the server. This request is passed internally with Express and Node from the Route to the fetchTrips() Controller/Model.

The Controller/Model takes the request through the Mongoose ODM to interact with the MongoDB server. MongoDB processes the request and responds with a callback to the Express/Nodejs server. Express/Nodejs sends a JSON response to the Angular component, which responds with a view back to the user’s browser.

Provide instances in the full stack process when you refactored code to improve functionality and efficiencies, and name the benefits that come from reusable user interface (UI) components.

Several instances of major refactoring occurred throughout the lifespan of the project. Initially, our Express/Node.js application fed data dynamically to the views via Handlebars templating. This enabled rapid prototyping by allowing us to quickly set up a backend in preparation for REST. Second major refactor occurred when migrating the static HTML to Handlebars template views into partials (header, footer, etc) and began reparating routing functionality from controllers in an MVC approach (one route per controller). As our views were still hardcoded, the next major step was to design the data schema model in Mongoose, and create the database in mongodb to work with the model. This entails connecting our application to the database, defining schemas and models, and finally adding test data to the database (can seed JSON test data with Mongo seed or mongoose). Then build the RESTful API with proper routes for get, push, post, delete functionality in Node/Express which will allow your application to interact with the database through Mongoose. The next step is to hook the database into the application . After creating the static application and an API exposing and interface to the database we want the application to talk to the API. The final stage was to augment the application with additional functionality by adding authentication system via crypto and Passport libraries, data validation on all layers (schema, server, front-end/Angular), and error handling methods for displaying error messages to users.

Testing

Methods for request and retrieval necessitate various types of API testing of endpoints, in addition to the difficulties of testing with added layers of security. Explain your understanding of methods, endpoints, and security in a full stack application.

In our application architecture, when a user posts their credentials to the server via an API, the server validates these credentials by using the database and returning a JSON web token to the browser, the browser can then save this token for reuse later. The difference between the mean stack approach and traditional server application, is that instead of storing a user session data on the server the data is instead stored in the browser client side. The JSON web token helps pass data around between the API on the server and the SPA in the browser. The JWT is also used by the server that generated the token to authenticate a user when it's returned in a subsequent request.

We can track user sessions for user logins by using JSON web tokens to securely call API endpoints. When calling an authenticated API endpoint, the browser will send the JWT along with the data, the server decodes the token to validate the user’s request. This can help by allowing certain database actions to be performed only by authenticated users thus overcoming the stateless API limitations and express. Once this token is sent to the API, the endpoint then decodes the token invalidates whether the user is permitted to make that call. When a user tries to log in the controller will call a method in userSchema.methods.validPassword in user.js. When a user has been found with a given email address the method will return true or false depending on whether the hashes match. This is called hashing and validating their credentials, using the same encryption on the password the user is trying to log in with to see whether it matches the stored value. We can restrict access to certain endpoints such as register and login to authenticated users by securing relevant API endpoints. To do this, we must limit access to the API endpoints by allowing only users who send a valid JWT with their request to call the new trip API. Additionally within the controller we should validate that the user exists and can create a new trip. Thus ensuring only registered users can add or edit trips. Express-jwt middleware can validate the supply JWT and then extract the payload data and add it to the request object for the controller to use. Express dash JWT exposes a function that can be passed an options object, which can be used to send the secret and also specify the name of the property you want to add to the request object to hold the payload. After configuring Passport, creating controllers, and registering the routes, update the router to add JWT authentication capabilities in /app\_api/routes/index.js. Update the routes that alter the database (add, update, and delete) by injecting the authentication middleware just added above. We use an authentication service in Angular to detect if a user is logged in, the Router module helps track history pages, and using information from the JWT a new method getCurrentUser() in authentication.service.ts will validate that a user is logged in by calling isLoggedIn() method. If the user is logged in, it gets the token by calling the getToken() method before extracting and decoding the payload and returning the email and user data.

Reflection

Full Stack has helped me to see how all of the interworking parts fit together to deliver the final product, from the Express Nodejs backend controllers routing data to the views to the MongoDB Mongoose functions returning res.json responses, and finally adding layers of security on client, server, schema-side. I’ve developed some essential asynchronous programming fundamentals further as well as interacting with and hooking into a database to feed into view templates. All these essential skills will be important to carry with me into my career.