**DevCamper Backend API Specifications**

Create the backend for a bootcamp directory website. The frontend/UI will be created by another team (future course). The html/css template has been created and can be used as a reference for functionality. All of the functionality below needs to be fully implmented in this project.

**Bootcamps**

* List all bootcamps in the database
  + Pagination
  + Select specific fields in result
  + Limit number of results
  + Filter by fields
* Search bootcamps by radius from zipcode
  + Use a geocoder to get exact location and coords from a single address field
* Get single bootcamp
* Create new bootcamp
  + Authenticated users only
  + Must have the role "publisher" or "admin"
  + Only one bootcamp per publisher (admins can create more)
  + Field validation via Mongoose
* Upload a photo for bootcamp
  + Owner only
  + Photo will be uploaded to local filesystem
* Update bootcamps
  + Owner only
  + Validation on update
* Delete Bootcamp
  + Owner only
* Calculate the average cost of all courses for a bootcamp
* Calculate the average rating from the reviews for a bootcamp

**Courses**

* List all courses for bootcamp
* List all courses in general
  + Pagination, filtering, etc
* Get single course
* Create new course
  + Authenticated users only
  + Must have the role "publisher" or "admin"
  + Only the owner or an admin can create a course for a bootcamp
  + Publishers can create multiple courses
* Update course
  + Owner only
* Delete course
  + Owner only

**Reviews**

* List all reviews for a bootcamp
* List all reviews in general
  + Pagination, filtering, etc
* Get a single review
* Create a review
  + Authenticated users only
  + Must have the role "user" or "admin" (no publishers)
* Update review
  + Owner only
* Delete review
  + Owner only

**Users & Authentication**

* Authentication will be ton using JWT/cookies
  + JWT and cookie should expire in 30 days
* User registration
  + Register as a "user" or "publisher"
  + Once registered, a token will be sent along with a cookie (token = xxx)
  + Passwords must be hashed
* User login
  + User can login with email and password
  + Plain text password will compare with stored hashed password
  + Once logged in, a token will be sent along with a cookie (token = xxx)
* User logout
  + Cookie will be sent to set token = none
* Get user
  + Route to get the currently logged in user (via token)
* Password reset (lost password)
  + User can request to reset password
  + A hashed token will be emailed to the users registered email address
  + A put request can be made to the generated url to reset password
  + The token will expire after 10 minutes
* Update user info
  + Authenticated user only
  + Separate route to update password
* User CRUD
  + Admin only
* Users can only be made admin by updating the database field manually

**Security**

* Encrypt passwords and reset tokens
* Prevent NoSQL injections
* Add headers for security (helmet)
* Prevent cross site scripting - XSS
* Add a rate limit for requests of 100 requests per 10 minutes
* Protect against http param polution
* Use cors to make API public (for now)

**Documentation**

* Use Postman to create documentation
* Use docgen to create HTML files from Postman
* Add html files as the / route for the api

**Deployment (Digital Ocean)**

* Push to Github
* Create a droplet - <https://m.do.co/c/5424d440c63a>
* Clone repo on to server
* Use PM2 process manager
* Enable firewall (ufw) and open needed ports
* Create an NGINX reverse proxy for port 80
* Connect a domain name
* Install an SSL using Let's Encrypt

**Code Related Suggestions**

* NPM scripts for dev and production env
* Config file for important constants
* Use controller methods with documented descriptions/routes
* Error handling middleware
* Authentication middleware for protecting routes and setting user roles
* Validation using Mongoose and no external libraries
* Use async/await (create middleware to clean up controller methods)
* Create a database seeder to import and destroy data