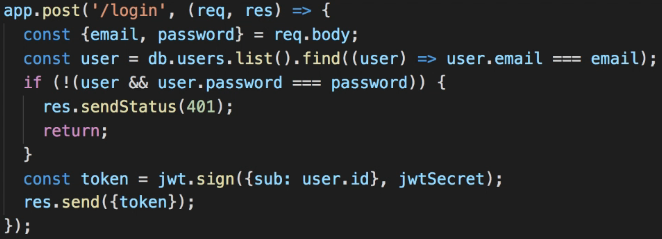
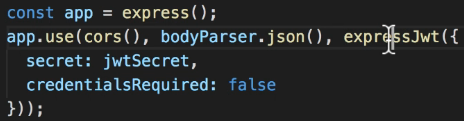
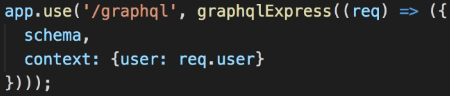
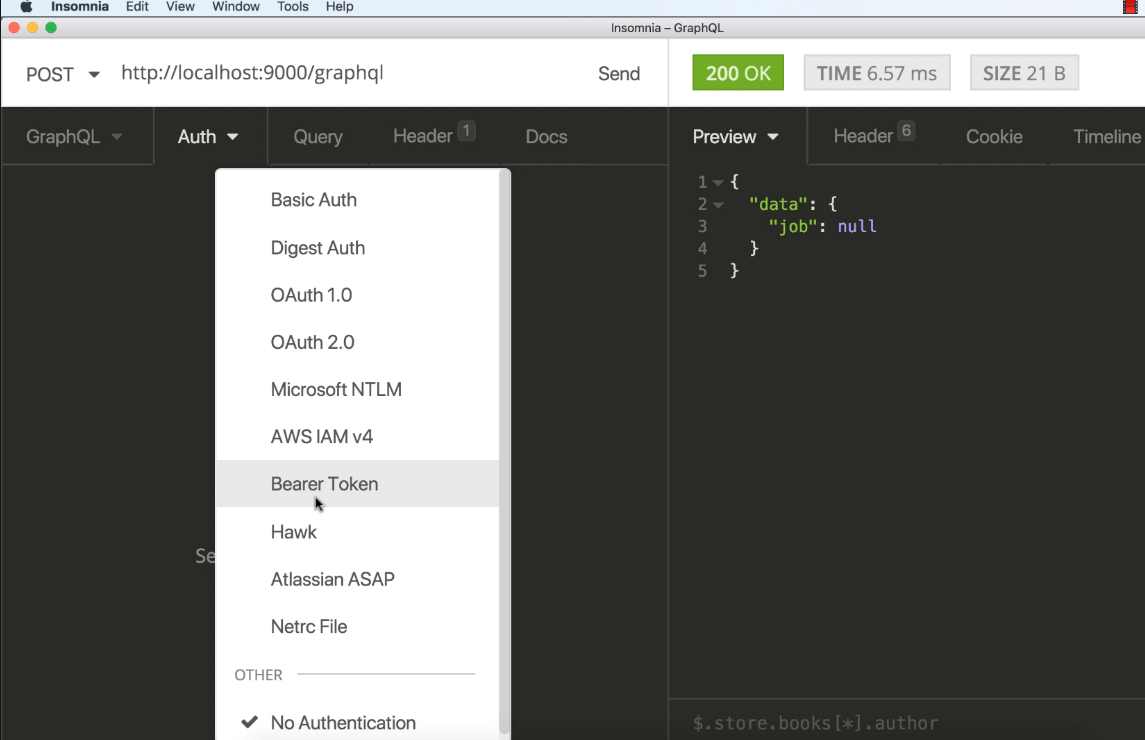
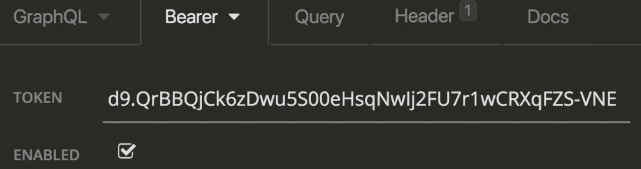
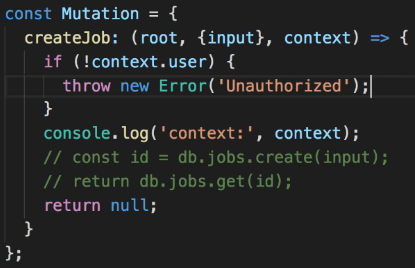
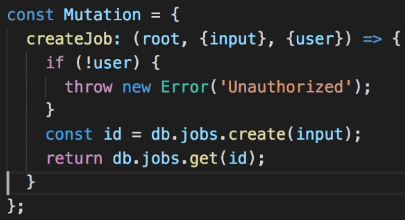
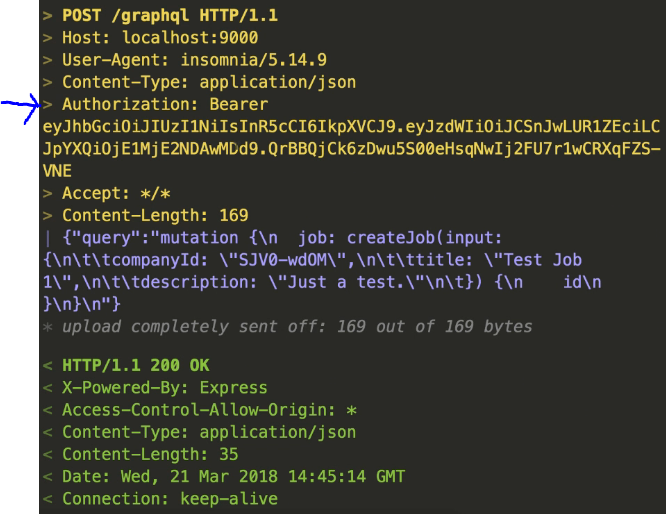
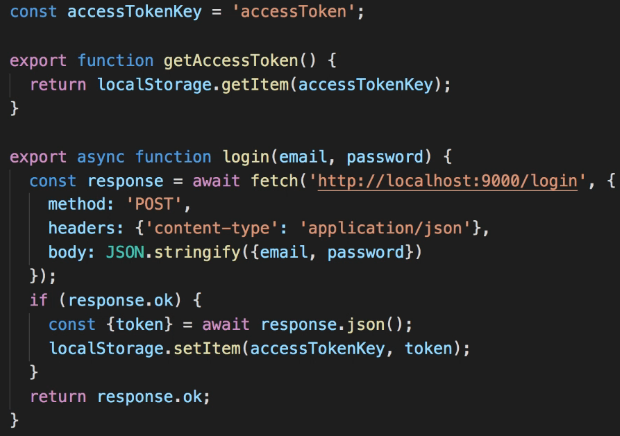
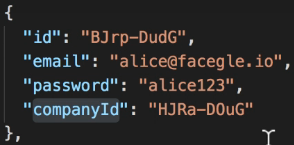
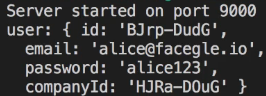
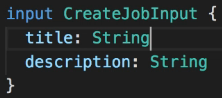
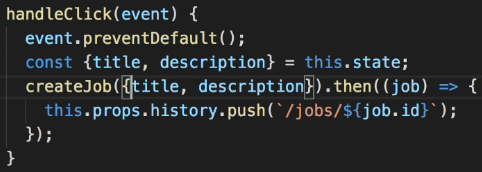
**Checking for Authentication in the Context**  
\* Before we fix the hard-coded ID, let’s add support for Authentication which is a separate feature but it’s also a pre-requisite for knowing which company to associate a new job.  
\* Support for authentication means that if we’re not logged in, then we shouldn’t be able to create a new job.  
\* In the Frontend if you’re not logged in, you don’t see the “Post Job” link but that’s not enough.  
=> **Somebody could call our GraphQL API directly like we can do from GraphiQL where we were able to create a job even without any authentication**.  
\* So we need to protect this createJob mutation somehow, we only want authenticated users to be able to post a job on our board.  
\* In our server.js there’s already some support for authentication - we know that we can already log in in the frontend, in this file you can see a “/login” route at the bottom. That’s what’s called when we log in front the frontend.  
  
\* The code gets email and password from the request, looks in the DB for a user with that email and if it can’t find that user or the password doesn’t match, it sends a 401 Unauthorized response, otherwise if the login is valid, it sends back an access token generated with JWT.  
  
\* This project also uses the expressJwt middleware to check if a user is authenticated.  
\* And that depends on whether the request contains the token generated by the login endpoint.  
\* It’s configured with credentialsRequired = false which means that by default you don’t have to be autheticated to access any route, but if you do send an access token, then the expressJwt middleware will check if it’s valid and extract your authetication details from the token.  
\* There are different ways to handle authentication in your server.  
\* express-jwt is a pretty popular option.  
<https://github.com/auth0/express-jwt>  
  
\* So in our code if we want to check if a user is authenticated, we can look at this req.user property.  
**context** => **3rd parameter passed to a resolver function**.  
=> **Can be used to access things that are not part of the GraphQL itself but are provided by our application**.  
=> **Context can contain pretty much whatever we want but it’s up to us to put something into the context in the first place**.  
\* Change:  
  
\* To:  
  
=> **This allows us to return a different Configuration Object depending on the request**.  
\* User will only be defined if we send a valid access token and the access token is what’s returned by the “/login” endpoint. It’s created using the JWT library.  
  
\* **What we should do with this token is send it back to the server in any subsequent request to prove that we are authenticated**.  
\* **The token is also available in the Local Storage because the frontend application saves it there after a successful login response**.  
\* Let’s see how we can send this token in our requests.  
\* Unfortunately, we can’t use GraphiQL for this because there’s no way to set HTTP headers in this tool.  
\* So I’m going to use another tool called **Insomnia**.  
=> **Insomnia is a nice app for testing HTTP Requests. It supports both REST and GraphQL**.  
  
\* Let’s paste our token here:  
  
\* Now if we send it, we can see in our console logs:  
  
\* **The user ID is the value of the “sub” property where “sub” stands for “subject” and it’s what identifies the user according to the JWT standard**.  
   
\* Throwing an error will cause GraphQL to return an error response of course.  
\* We can use destructuring.  
  
\* Our createJob mutation can now only be invoked by authenticated users.  
\* We do that by looking at the user property available in the context.  
\* If user is set, then we can proceed, otherwise we throw an error.  
\* It’s important to remember that we need to set up the context object where we create the “/graphql” endpoint in the graphqlExpress function configuration.  
\* If you’re using a different authentication mechanism, you may have to do something slightly different but **the main point is that you can use the Context to provide whatever application-specific objects you need, to your GraphQL code**.

**Restrict Users to Create Jobs for their Company**  
\* We added a security check to our server code to make sure that only autheticated users can create a new job.  
\* We’ve also seen how to provide authentication details when we make a request - we need to pass this token that was returned by the “login” request.  
  
\* In our client code, we’ll need to set the same header if we want our request to be authenticated.

\* **Let’s pretend we have the isLoggedIn() and getAccessToken() functions**.  
  
\* In auth.js:  
  
  
\* So in requests.js we can just import them.  


**Sending Authentication in Client Requests**  
  
\* What we’d like to do when we create a job is to set the company for the new job to be the same company of the authenticated user.  
\* The only problemis that if we look at the user object logged to the console, we don’t have the companyId there. That’s because the “user” property we put into the context right now is the req.user set by expressJwt which only contains information decoded from the token.  
=> It would be better if we could pass a user object loaded from the database instead.

  
\* There’s a potential issue here - there’s no guarantee that req.user will be set.  
=> If a user is not logged in on the frontend, then the request will not include the access token and in that case req.user will be undefined.  
\* As a simple workaround we can check if req.user is defined.  
  
  
\* At this point, since we set the company based on the autheticated user, we don’t need the companyId to be passed in as a GraphQL argument anymore.  
=> So we should update our schema.  
  
\* Now our resolver code is working fine, each new job will have the same companyId as the user that created it.  
\* The last thing we need to do is update our client project - removing the hard-coded companyId and not passing it to the createJob anymore.  
  
\* So the client is no longer passing a companyId when creating a job, instead, the server automatically assigns a companyto the new job based on the authenticated user.

\* That’s more secure because it means each user can only create jobs for their own company.