Deploying Blazor WASM to Azure Static Web Apps

Let's start off by doing a standalone deployment. Instead of using a classic option like IIS, we'll use the Azure Static Web Apps service which is a perfect service to host our Blazor WebAssembly static files on. Azure Static Web Apps can do much more than hosting Blazor WebAssembly, but we'll use it to do exactly that. In order to follow along, you'll need an Azure account and a GitHub account, since GitHub actions are used to build the project before deploying it to the Azure. To demonstrate the process, we'll use the client project from the **module12-end** branch, but you can use your own repository to play around. You can fork our own repository, or create a completely new one. Alright, let's start things off with logging into the Azure portal and searching for the static web apps service first. Let's click on that and go to the azure web apps service page. Once we're on the page, we'll go to "New" and we'll get a new static web app creation wizard. On this form, we want to select a Subscription, create or select a resource group, give our app a name, something like **module12-end** or anything you like, and pick a region closest to us. After that, we need to sign in with our GitHub account to be able to access and work with our repository. After we're prompted, we need to accept the authorization… confirm the password if offered… and we can access the repository now... Great! Now we can see three additional fields on our form… Let's pick the Organization... and then the repository… and finally let's pick the branch we want… Okay, now the deployment details are set up. Next on, we want to configure GitHub action details… Let's pick a Blazor preset… enter **BlazorProducts.Client/BlazorProducts.Client** as our app location since that's where our csproj file resides… we don't care much about API since we don't have an API in our project, and as an output location, we can leave wwwroot. We can do a quick preview of the workflow file to see if everything is in order… But we don't want to worry about this file too much right now since we'll be able to change it if needed in our GitHub actions panel. Alright, let's go to the Review + Create and check out our details quickly once more… and then click Create… This should trigger the build and we can see that it says Deployment is in Progress… If we're quick enough, we can see that the GitHub action is triggered… GitHub action actually builds the project using docker engine, and it practically publishes our artifacts and deploys them to Azure Static Web Apps resource afterward

if the build succeeds. Ok great, once that the build is finished, we can go to our resource... And navigate to the Url we've been assigned… And once the page loads, we can see our application running in a production environment which is great! But if we try to go to the products page… We'll see the application trying to fetch the data… and failing… That's completely okay since the application is still trying to fetch the API on the localhost:8081, which was our production URL. There's one more thing we need to take care of. Let's see what happens if we go to the products page… and try to reload the application… We get a 404: Not Found message from Azure. If you remember how we configured the routes in the last video when we hosted the application in ASP.NET Core, we had to add a fallback route that leads to index.html since this is a single page application and every route needs to serve the index.html file. Since we don't rely on the ASP.NET Core server anymore, we need to configure routes again somehow. We can do that by adding the **routes.json** file to the wwwroot of the application… and then adding the following lines…

```
{
  "routes": [
    {
      "route": "/*",
      "serve": "/index.html",
      "statusCode": 200
    }
  ]
}
```

Once we commit the changes, and the GitHub actions publishes and deploys our application again, this file will indicate that all the routes fallback to index.html again… And to test this out, we can go to the products URL again… and then refresh the page… and as we can see we don't get a 404 not found anymore. Excellent. Now if you want to see what's happening behind the scenes and maybe find out why your build fails, you can go to your repository on GitHub, and then to the Actions tab… and pick the latest action that is related to your Azure Static Web apps workflow. Inside it, you can find all kinds of useful information, including the fail reason and all the build logs that can help you debug the problem. Alright, let's wrap this video

up. Since this is a standalone deployment, our API is not available and we can see that Products haven’t been fetched. That’s completely fine because in the next video we’ll deploy our API too, and hit the endpoints once it’s been deployed.