MIMOS Angular Workshop

1 Labs

1.1 TypeScript

We will compile the Typescript files using `tsc` and execute the resulting Javascript files using Node.
https://www.keycdn.com/blog/typescript-tutorial#Part-2-Compiling-to-JavaScript

The sample files are in `tsdemo`

We will run the compiler in watch mode:

```
tsc -w *.ts
```

to monitor all the Typescript files in our directory and trigger recompilation on changes.

More details on using the Typescript compiler:

https://blog.angularindepth.com/configuring-typescript-compiler-a84ed8f87e3
https://www.typescriptlang.org/docs/handbook/compiler-options.html

1.2 Intro: Basic app with binding

1.2.1 stock-market

This application is built from scratch

Step 1: Creating a new Angular app

From command prompt:

```
ng new stock-market
```

```
ng serve
```

Exit and rerun `ng serve` from within the terminal in VS-Code

Step 2: Generating a component

In `src/app`

```
ng generate component stock/stock-item
```

Step 3: Customizing template content

`app.component.v1.html`

Step 4: Definition of various instance variables

`stock-item.component.v1.ts`

Step 5: Creating customized CSS for the component

`stock-item.component.v1.html`
`stock-item.component.v1.css`

Step 6: One way data binding

`stock-item.component.v2.html`

Step 7: Adding additional instance variable

`stock-item.component.v2.ts`

Step 8: Adding additional classes to the CSS

`stock-item.component.v2.css`

Step 9: Property binding

`stock-item.component.v3.html`

Experiment with changing the values of `price` and `previousPrice` in `stock-item.component.ts` to change `positiveChange` and observe the effect in `stock-item.component.html`

Step 10: Adding a new function

`stock-item.component.v3.ts`

Step 11: Event binding

`stock-item.component.v4.html`

Click to see button become disabled

Step 12: Using $event in the template

`stock-item.component.v5.html`

Step 13: Accessing event in the component

`stock-item.component.v4.ts`

Step 14: Creating a data model

In `src/app`

```
ng generate class model/stock
```

`stock.v1.ts`

Step 15: Using the data model in the component

`stock-item.component.v5.ts`

Step 16: Modifying template to reflect this use

`stock-item.component.v6.html`

1.3 Directives

1.3.1 stock-market-directives

Start from the code base in `stock-market/stock-market-final`

Step 1: Adding new classes to the CSS

`stock-item.component.v1.css`

Step 2: Create object in component

`stock-item.component.v1.ts`

Step 3: Using NgClass directive

`stock-  item.component.v1.html`

Experiment with instantiating Stock object instance with different values for previousPrice and price to view the effect on the template.

Step 4: Create style object based on properties

`stock-item.component.v2.ts`

Step 5: Using NgStyle directive

`stock-  item.component.v2.html`

Experiment with instantiating Stock object instance with different values for previousPrice and price to view the effect on the template.

Step 6: Adding / removing one particular class/style

`stock-  item.component.v3.html`

Step 7: Using NgIf Directive

`stock-  item.component.v4.html`

Clicking on the Add to Favourite button should remove it

Step 8: Modify component to use an array of Stocks

`stock-item.component.v3.ts`

Step 9: Using the NgFor directive

`stock-  item.component.v5.html`

Step 10: Add function to track individual items

`stock-item.component.v4.ts`

Step 11: Using modified NgFor directive

`stock- item.component.v6.html`

1.4 Working with components

1.4.1 component-stuff

Start from the code base in `stock-market\stock-market-final`

Step 1: Using an inline template

`stock-item.component.v1.ts`

Step 2: Using inline styles

`stock-item.component.v2.ts`

Step 3: Style encapsulation

`app.component.v1.css`

Step 4: Style encapsulation continued

`app.component.v1.ts`

Step 5: Input decorator in child component

`stock-item.component.v3.ts`

Step 6: Creating object in parent component

`app.component.v2.ts`

Step 7: Passing input data through the template

`app.component.v1.html`

Step 8: Output decorator in child component

`stock-item.component.v4.ts`

Step 9: Modify template for new function call

`stock-item.component.v1.html`

Step 10: Add trigger method to parent component

`app.component.v3.ts`

Step 11: Passing output data through the template

`app.component.v2.html`

Step 12: Creating ChangeDetectionStrategy in child component

`stock-item.component.v5.ts`

Step 13: Adding a button in the child template

`stock-item.component.v2.html`

Step 14: Adding buttons in the parent template

`app.component.v3.html`

Step 15: Functions to test in parent component

`app.component.v4.ts`

Experiment with pressing all buttons

Step 16: Component life cycle hooks on parent component

`app.component.v5.ts`

Step 17: Component life cycle hooks on child component

`stock-item.component.v6.ts`

Step 18: Using ngContent for content projection

`stock-item.component.v3.html`

Step 19: Add test method in parent component

`app.component.v6.ts`

Step 20: Modify parent template for content projection

`app.component.v4.html`

1.4.2 angular-reddit

Final code base in `angular-reddit`

1.5 Forms

1.5.1 forms-basic

Start from the code base in `forms-basic\forms-basic-start`

Step 1: Import appropriate modules

`app.module.v1.ts`

Step 2: Create a new component

In `src/app`

```
ng generate component stock/create-stock
```

Step 3: Modify create-stock

```
create-stock.component.v1.ts
```

Step 4: Edit new template

```
createstock.component.v1.html
```

Step 5: Modify parent template to include new template

```
app.component.v1.html
```

Fill in the text form and note the changes

Step 6: Using NgModel directive

```
createstock.component.v2.html
```

Step 7: Using banana-in-the-box syntax

```
createstock.component.v3.html
```

Step 8: Modifying data model and components

```
stock.v1.ts
```

```
create-stock.component.v2.ts
```

```
app.component.v1.ts
```

Step 9: Modifying template for the component

```
createstock.component.v4.html
```

Step 10: Modification to use NgFor for select drop-down box

`createstock.component.v5.html`

`create-stock.component.v3.ts`

Step 11: Modification of CSS for color scheme for validation

`create-stock.component.v1.css`

Step 12: Modification of template for validation

`createstock.component.v6.html`

Step 13: Modification of CSS for color scheme for validation

`create-stock.component.v2.css`

Step 14: Change to component to do logging

`create-stock.component.v4.ts`

Step 15: Using template variables

`createstock.component.v7.html`

Step 16: Modifying component to work with FormGroups

`create-stock.component.v5.ts`

Step 17: Modifying template to work with FormGroups

`createstock.component.v8.html`

1.5.2 forms-reactive

Start from the code base in `forms-basic\forms-basic-final`

Step 1: Import ReactiveFormsModule

`app.module.v1.ts`

Step 2: Using formControl binding in template

`create-stock.component.v1.html`

Step 3: Changing component to support template

`create-stock.component.v1.ts`

Step 4: Modifying template to obtain all fields

`create-stock.component.v2.html`

Step 5: Modifying component to support FormGroup

`create-stock.component.v2.ts`

Step 6: Modifying component to support FormBuilders

`create-stock.component.v3.ts`

Step 7: Modify template for error messages

`create-stock.component.v3.html`

Step 8: Modify template to simulate resetting and loading to a server

`create-stock.component.v4.html`

Step 9: Change classes to fit these changes

`create-stock.component.v4.ts`

`stock.v1.ts`

`app.component.v1.ts`

Step 10: Update model to include array

`stock.v2.ts`

Step 11: Update component to use FormArray

`create-stock.component.v5.ts`

Step 12: Update CSS for separation

`create-stock.component.v1.css`

Step 13: Modify template to include array

`create-stock.component.v5.html`

1.6 Services

1.6.1 services-basic

Start from the code base in `forms-basic\forms-basic-final`

Step 1: Add additional buttons to template

`stock-item.component.v1.html`

Step 2: Create Stock-list component

In `src/app`

```
ng generate component stock/stock-list
```

Step 3: Modify new component

`stock-list.component.v1.ts`

Step 4: Modify template of new component

`stocklist.component.v1.html`

Step 5: Modify parent component and template

`app.component.v1.ts`

`app.component.v1.html`

Step 6: Create a stock service

In `src/app`

```
ng generate service services/stock
```

Step 7: Edit stock service to return dummy data

`stock.service.v1.ts`

Step 8: Registering service as provider

`app.module.v1.ts`

Step 9: Using service in stock list component

`stock-list.component.v2.ts`

Step 10: Modify template to show service

`create-stock.component.v1.html`

Step 11: Modify component to use service

`create-stock.component.v1.ts`

Step 12: Creating a message service and registering it as a provider

In `src/app`

```
ng generate service services/message
```

`app.module.v2.ts`

Step 13: Update the message service

`message.service.v1.ts`

Step 14: Modify the main component and template to use it

`app.component.v2.ts`

`app.component.v2.html`

Step 15: Modify child component to use the same service

`create-stock.component.v2.ts`

Step 16: Modify template to show the service

`create-stock.component.v2.html`

Step 17: Adding provider at a child component level

`create-stock.component.v3.ts`

1.7 Observables

1.7.1 observables-basic

Start from the code base in `observables-basic\observables-basic-start`

Step 1: Add observables to service

`stock.service.v1.ts`

Step 2: Change components to read directly from observable

`stock-list.component.v1.ts`

`create-stock.component.v1.ts`

Step 3: Further simplification to use observable

`stock-list.component.v2.ts`

Step 4: Modification of template to fit this

`stock-list.component.v1.html`

1.8 HTTP

1.8.1 http-basic

Start from the code base in `observables-basic\observables-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

to start the back-end server. This basic server exposes 3 APIs:

- GET on `/api/stock` to get a list of stocks
- POST on `/api/stock` with the new stock as a body to create a stock on the server
- PATCH on `/api/stock/:code` with the stock code in the URL and the new favorite status in the body of the request, to change the state of favorite for the particular stock.

Type this URL into the address bar of the browser to test out the GET API:

```
http://localhost:3000/api/stock
```

Confirm the return response of this JSON document:

[{"name":"Test Stock Company","code":"TSC","price":85,"previousPrice":80,"exchange":"NASDAQ","favorite":false}, {"name":"Second Stock Company","code":"SSC","price":10,"previousPrice":20,"exchange":"NSE","favorite":false}, {"name":"Last Stock Company","code":"LSC","price":876,"previousPrice":765,"exchange":"NYSE","favorite":false}]

Step 1: Add a dependency on HttpClientModule

`app.module.v1.ts`

Step 2: Change component to make HTTP calls

`stock.service.v1.ts`

Step 3: Change data model to interface

`stock.v1.ts`

Step 4: Modifying component and template to use new data model

`stock-list.component.v1.ts`

`stock-list.component.v1.html`

Step 5: Modifying another component and template

`stock-item.component.v1.ts`

`stock-item.component.v1.html`

Step 6: Modify creation of stock

`create-stock.component.v1.ts`

Step 7: Create proxy file

Create a file `proxy.conf.json` in the main project folder and populate it with:

```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false
  }
}
```

Step 8: Serve the app with reference to the proxy

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

1.8.2 http-advanced

Start from the code base in `http-basic\http-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

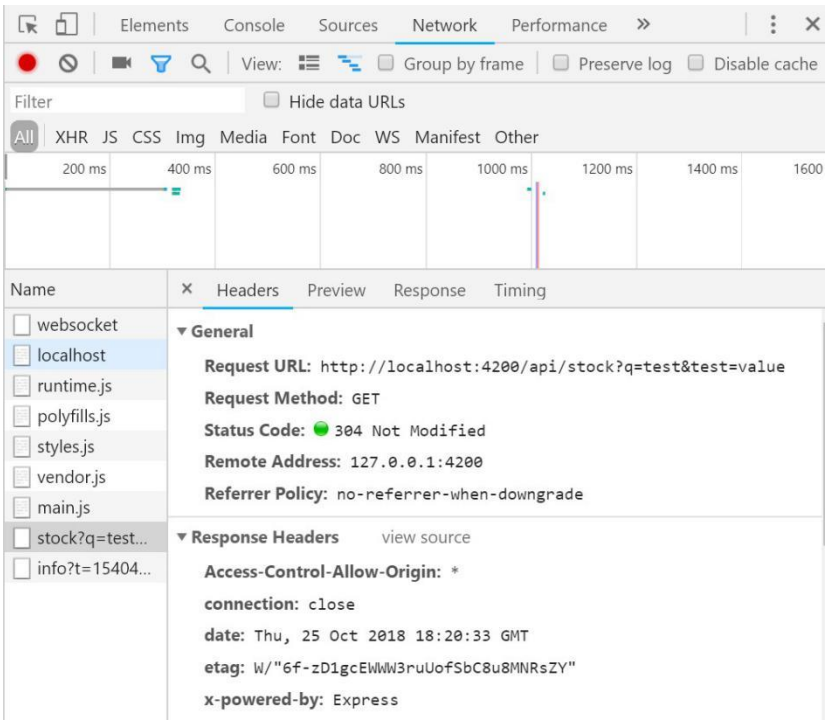
In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Adding HTTP headers

`stock.service.v1.ts`

After reloading, select Network and in the Headers tab for the stock connection, you should be able the new request URL and header. The server in this case will only provide a response of a single stock corresponding to those URL parameters (`q=test&test=value`)



Step 2: Modify request to add observe property

`stock.service.v2.ts`

Step 3: Modify component to make calls to the APIs

`stock-list.component.v1.ts`

1.8.3 http-interceptor

Start from the code base in `http-basic\http-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Generate a service for authorization and register it

In `src/app`
```
ng generate service services/auth
```

`app.module.v1.ts`

Step 2: Add property to Auth service

`auth.service.v1.ts`

Step 3: Add HTTP API call

`stock.service.v1.ts`

Step 4: Change component to add extra buttons

`stock-list.component.v1.ts`

Step 5: Change template to accommodate new buttons

`stocklist.component.v1.html`

Step 6: Create the interceptor

Create:
`src/app/services/stock-app.interceptor.ts`

Step 7: Add interceptor to main module

`app.module.v1.ts`

Step 8: Adding functionality to the interceptor

`stock-app.interceptor.v2.ts`

1.8.4 observables-extra

Start from the code base in `http-basic\http-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Edit template, show number of stocks

`stocklist.component.v1.html`

Check that 2 calls are made to the server

Step 2: Modify template to make calls

`stock-list.component.v1.ts`

Check that 1 call is made to the server

Step 3: Add code to search for stocks based on query string

`stock.service.v1.ts`

Step 4: Change template to make updated call

`stock-list.component.v2.html`

Step 5: Modify component to reflect change

`stock-list.component.v2.ts`

Reload and note in the network component that a HTTP call is made for every key stroke

Step 6: Augment component with observable operators

`stock-list.component.v3.ts`

1.8.5 portfolio

Start from the code base in `portfolio\portfolio-start`

Do `npm install` first. You will get a bunch of warning messages as this uses a few deprecated modules and an older version of Angular is being used.

Then do `ng serve`.

Step 1: Modify the existing account service

`account.service.v1.ts`

Step 2: Modify app component to use service

`app.component.v1.ts`

Step 3: Register service in the module

`app.module.v1.ts`

Step 4: Modify component to use service

`stocks.component.v1.ts`

Step 5:

`investments.component.v1.ts`

Step 6: Include top bar in template

`app.component.v1.html`

Toolbar should now be populated with values from Account Service

Step 7: Create a class with static values for configuration

`config.service.v1.ts`

Step 8: Use configuration class to set values in main

`main.v1.ts`

Step 9: Use HTTP client in stock service

`stocks.service.v1.ts`

Step 10: Register service in the module

`app.module.v2.ts`

Step 11: Use stock service in app component

`app.component.v2.ts`

Step 12: Include remaining components in template

`app.component.v2.html`

You should now be able to see the full stock display panel

Step 13: Use the HTTP interceptor

`interceptor.service.v1.ts`

Step 14: Import token in app module

`app.module.v3.ts`

Step 15: Implement local storage

`local-storage.service.v1.ts`

Step 16: Use local storage in account service

`account.service.v2.ts`

Step 17: Do initialization from app component

`app.component.v3.ts`

You should be able to see the full performance of the main dashboard view by now

Step 18: Add service for alerts

`alert.service.v1.ts`

Step 19: Use service in component

`alert.component.v1.ts`

Step 19: Register provider with module

`app.module.v4.ts`

Step 20: Include alert in main template

`app.component.v3.html`

Step 21: Modify app component to use the alert service

`app.component.v4.ts`

Step 22: Modify account service to use the alert service

`account.service.v3.ts`

You should now be able to see the alert corresponding to buying and selling actions

1.9 Routing

1.9.1 Routing

Start from the code base in `routing\routing-start`

In the folder `routing\second-server`
run:

```
npm install
node index.js
```

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Setting up index.html

`index.v1.html`

Step 2: Generate routing module

In `src/app`

```
ng generate module app-routes --flat --routing
```

Step 3: Update routing module

`app-routes-routing.module.v1.ts`

Step 4: Linking route to main module

`app.module.v1.ts`

Step 5: Modify main template to load components for routes

`app.component.v1.html`

Step 6: Modify main template to allow navigation

`app.component.v2.html`

When loading the app, notice that you are not able to get the list of stocks from the stock list section yet as this is a protected section, only accessible after a valid login

Step 7: Modify CSS for main template

app.component.v1.css

Step 8: Provide default route for initial load

app-routes-routing.module.v2.ts

Step 9: Provide default route for incorrect URL

app-routes-routing.module.v3.ts

Step 10: Add new route

app-routes-routing.module.v4.ts

Step 11: Update new component

stock-details.component.v1.ts

Step 12: Modify registration component

register.component.v1.ts

Step 13: Modify login component

login.component.v1.ts

Verify that you can register and login using an appropriate user name / password combination. Also incorrect user name / password combinations for login or existing user names for registration are flagged according. After successful login, you are redirected to the stock list route, where the stocks are finally retrieved.

Step 14: Modify template to navigate to specific stock

stock-item.component.v1.html

Now you should be able to display a specific stock by clicking on one of the stocks in the list

Step 15: Modify component to pass query params

login.component.v2.ts

Step 16: Modify component to read query params

`stock-list.component.v1.ts`

Step 17: Modify template to include button

`stock-list.component.v1.html`

Step 18: Modify component to use observable

`stock-list.component.v2.ts`

Step 19: Generate and update authentication guard

In `src/app`

```
ng generate guard guards/auth
```

`auth.guard.v1.ts`

Step 20: Update main module with the guard

`app.module.v2.ts`

Step 21: Add auth guard to the routing module

`app-routes-routing.module.v5.ts`

Check that trying to navigate directly to the stock list or create stock page will end up redirecting you to the login page.

Step 22: Generate and update deactivation guard

In `src/app`

```
ng generate guard guards/CreateStockDeactivate
```

`create-stock-deactivate.guard.v1.ts`

Step 23: Update main module with the guard

`app.module.v3.ts`

Step 24: Add deactivation guard to the routing module

`app-routes-routing.module.v6.ts`

Reload the app, log in and navigate to the create stock page, and then try clicking any of the links at the top. At that point, you should see the confirmation asking whether you really want to navigate away.

Step 25: Generate a Resolver service

In `src/app`

```
ng generate service resolver/stock-load-resolver
```

Step 26: Update Resolver service

`stock-load-resolver.service.v1.ts`

Step 27: Update main module with the guard

`app.module.v4.ts`

Step 28: Add deactivation guard to the routing module

`app-routes-routing.module.v7.ts`

Step 29: Modify component to prefetch information

`stock-details.component.v2.ts`

1.9.2 Music

Final code base in `routing\music`

You will need to generate a new API key before running this app.
Go to `src/environments`, rename the existing Typescript file containing the hardcoded API key to `old-spotifyApiKey.ts`

Go to scripts, and execute the Javascript file there to generate a new API key in `src/environments`

```
node spotifyKey.js
```

Return back to the root project folder and start the app in the usual way:

```
ng serve
```

1.10 Testing

1.10.1 unit-test-basic

Start from the code base in `unit-test-basic\unit-test-basic-start`

Step 1: Initial app unit test

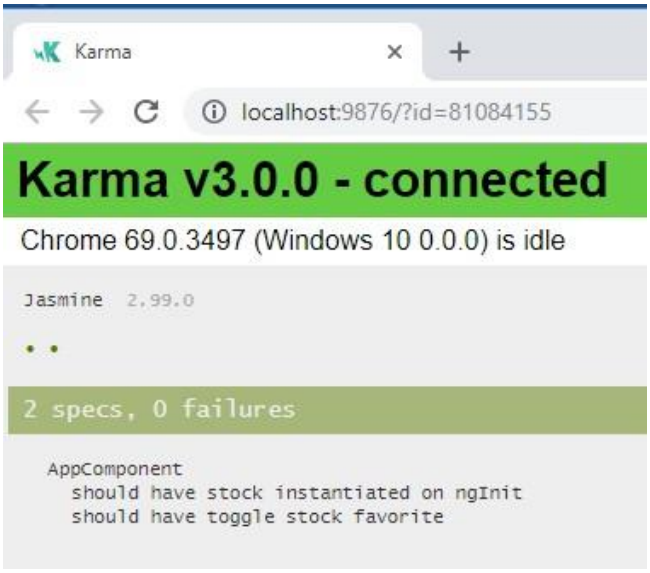
In `src\app`, create

`app.component.spec.ts`

In the main project folder, run test with:

```
ng test
```

A successful test should give a result similar to below:



Step 2: Creating an Angular aware test

Create

`stock-item.component.spec.ts`

Step 3: Testing component interaction

`app.component.spec.v2.ts`

1.10.2 unit-test-service

Start from the code base in `observables-basic\observables-basic-start`

Step 1: Original test: (no need for modification)

`stock.service.spec.ts`

Step 2: Add test for adding and fetching a list of stocks

`stock.service.spec.v2.ts`

Step 3: Modify component to test with real service

`stock-list.component.spec.ts`

Step 4: Modify component to test with mock calls

`stock-list.component.spec.v2.ts`

Step 5: Modify component to test with fake service

`stock-list.component.spec.v3.ts`

1.10.3 unit-test-async

Start from the code base in `observables-basic\observables-basic-final`

Step 1: Testing async

`create-stock.component.spec.ts`

Step 2: Testing using fake async

`create-stock.component.spec.v2.ts`

1.10.4 unit-test-http

Start from the code base in `http-basic\http-basic-final`

Step 1: Test initialization logic of fetching using HTTP GET

`stock-list.component.spec.ts`

1.10.5 Music

Final code base in `routing\music`

We integrate PhantomJS here with Karma to perform headless testing of web applications, suitable for as part of a continuous integration system.

To run unit tests
```
ng test
```

To run end to end test
```
ng e2e
```

1.10.6 unit-test-forms

Final code base in `unit-test-forms\forms`

To run unit tests
```
ng test
```

To run end to end test
```
ng e2e
```

1.11 Deploying

Once we have created a production build, you can upload the bundled assets and scripts generated from the build to a public server. Typically, this will be server within your company infra or an external cloud server instance. Many platforms now offer serverless deployment which greatly simplifies the process of deploying onto a public cloud by abstracting away the lower-level details of server management. An example is: https://zeit.co/ which we will use:

Install Zeit Now for deployment with:

```
npm install -g now
```

We will use the `angular-reddit` app to demonstrate this process. In the main project folder, type

```
ng build --prod
```

Then switch to dist folder in the command prompt and deploy with now:

now

Enter a valid email address to complete the initial registration. Check the inbox for an email and click on the link to complete verification. A confirmation message is displayed at the prompt. Type

now

again to deploy the app and obtain the public URL to access it (this will be in the form https://xxxxx.now.sh)