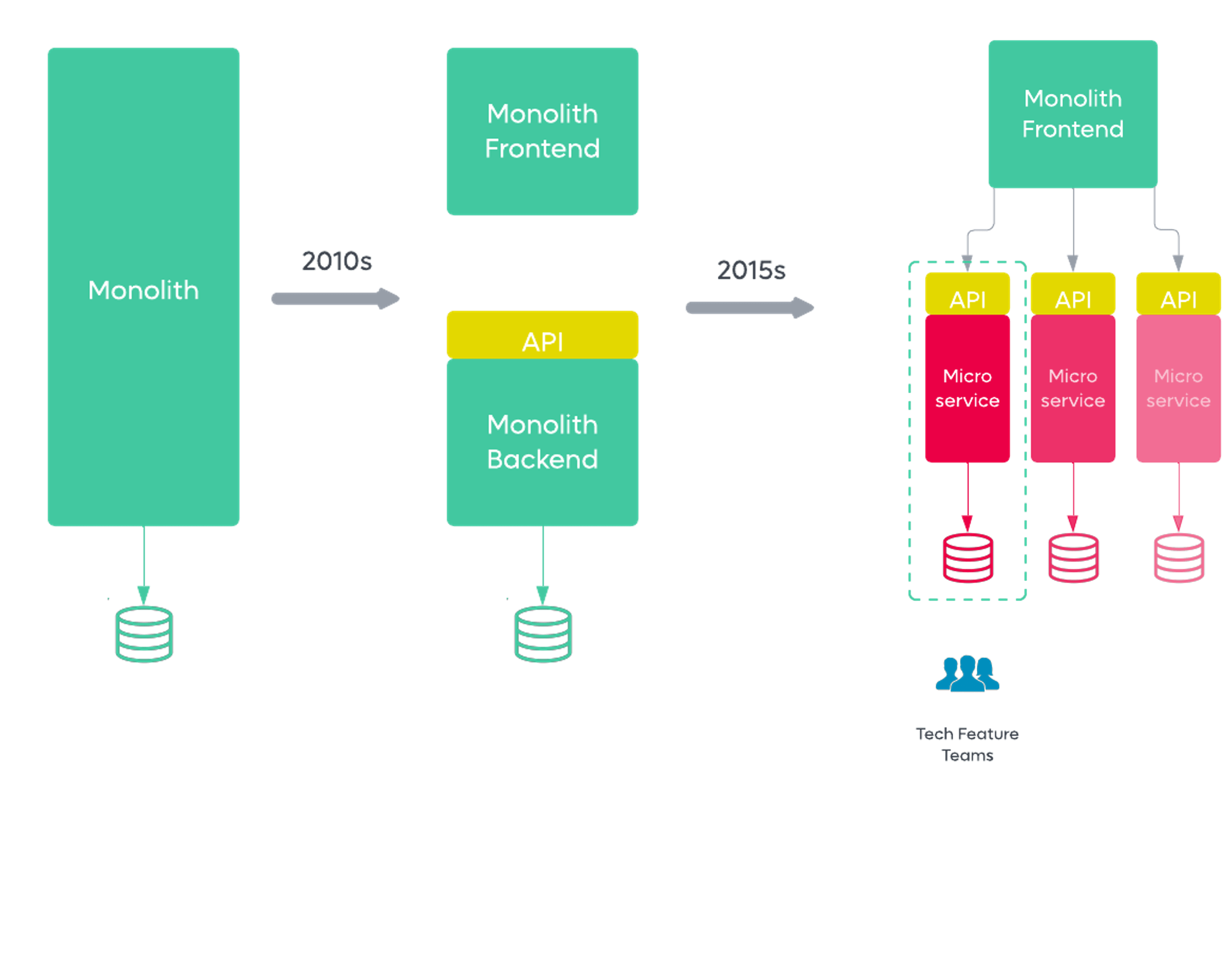
Introduction

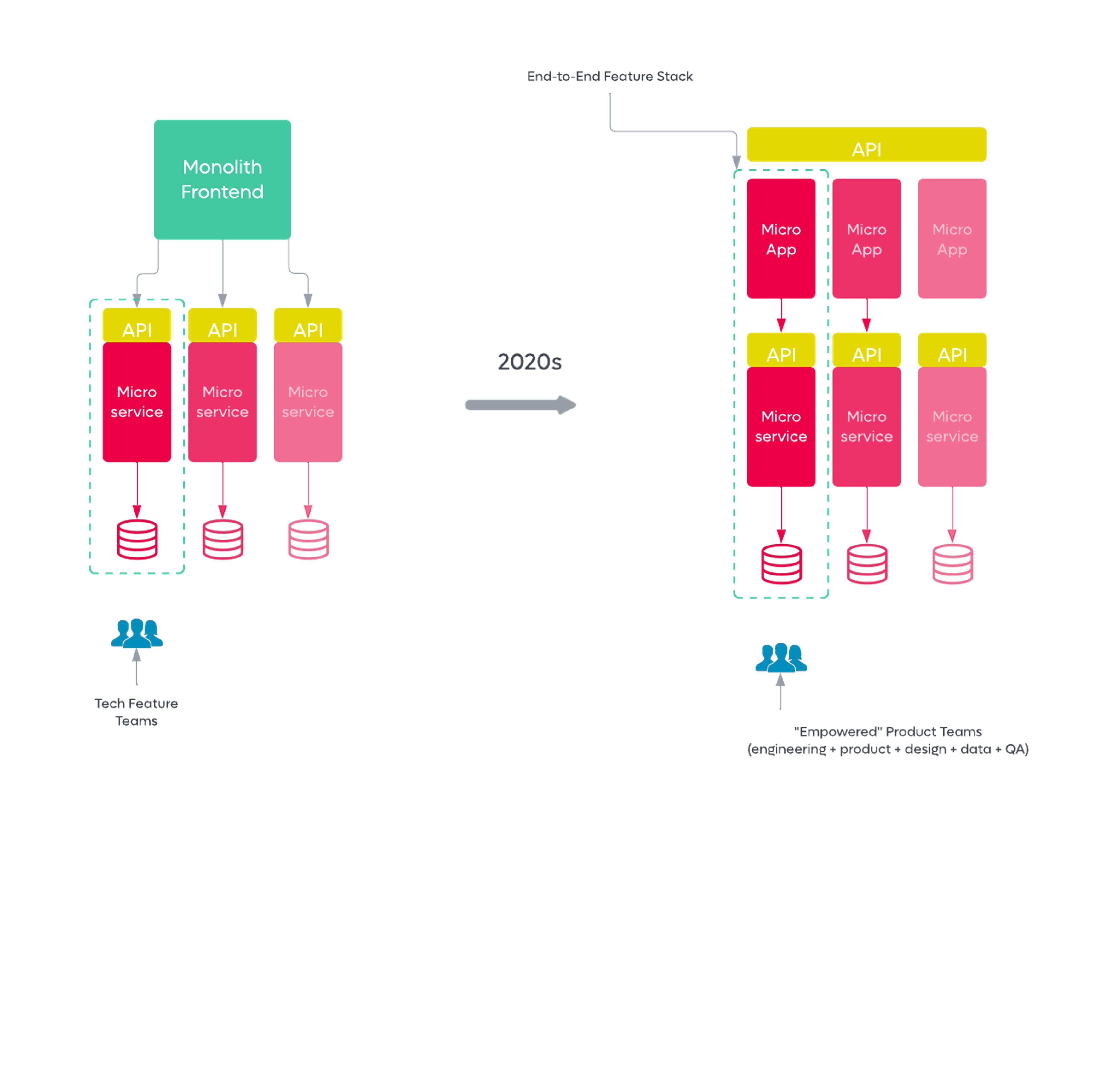
The purpose of this document is to create basic understanding of how angular works and web component configure in angular module. How expo-ui interacting with booking-ui as web component.

Overview:

In monolithic architecture, applications have huge components and complex logic. Therefore, maintaining or upgrading such applications is challenging. Moreover, integrating a new team can be a difficult task.



A better solution would be breaking down a vast application into smaller independent applications known as micro-frontends.



Micro-frontends are easier to manage, scale and assign to separate teams. This article introduces the reader to micro-frontends and their advantages compared to monolithic applications. We will use a demo application to illustrate how micro-frontends works.

Features of Micro-frontends

* Each frontend in the micro-frontend application is designed to solve a specific problem or provide a unique feature.
* Independent team members are assigned to implement a frontend in an application.
* The frontends cannot share the logic since they are independent of each other.
* A given team can manage a given frontend.

Advantages of micro-frontends

* **Applications are small** - In micro-frontends, an extensive application is split into small sections, pages, or even features. This makes the entire application small hence it would not take more storage space and memory.
* **Independent** - Since the applications are split and developed by different teams, the teams can work independently. In addition, if one application module is not working, it does not affect the entire application.
* **Easy to develop and deploy** - The applications that have been distributed over a team are easy to develop as the independent team members works on their part. They are also deployed independently with ease compared to a huge application.
* **Easy to test** - Large application requires a lot of unit testing before deployment, thereby increasing deployment time. However, unlike huge applications, unit testing is done independently in the micro-frontend since each application requires fewer unit tests making deployment much faster.
* **Less development time** - As mentioned earlier, micro-frontends require less development time than large applications as the separate teams work on their application independently instead of one team working on the application.
* **Easy CI/CD** - The application can be integrated and deployed independently hence making the CI/CD easier. Suppose a part of the application develops a bug or an update is required. In that case, it is easy to fix and perform repairs on the specific part of the application without interfering with the entire application.
* **Independent stacks and versions** - An application can have different versions of the same stack. This means that some teams can develop and test newer versions of the same stack in an application.
* **No shared code** - Large applications share the code to use the same functionality of some features. However, this comes at a cost in case of a bug that can entirely bring the application down. Also, the interdependencies between the modules can bring more problems as the application grows huge. This cannot happen in the micro-frontends as this architecture does not share code.

How Angular App Starts?

It would be a theory but again an important one if you want to master Angular.

When you open up your angular project you might have noticed a file sitting at the project root - angular.json

It is a very important file and contains several important configuration information to run the angular application.

architect -> build -> options , you would see a main node

Booking-UI:

A computer screen shot of a program code

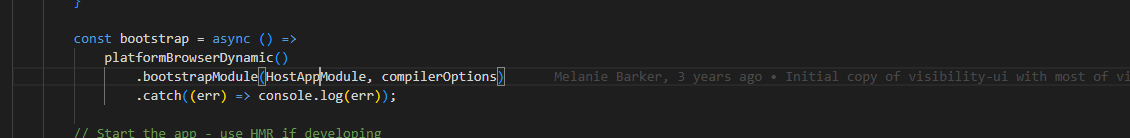
Description automatically generated

EXPO UI:



Once Angular found the file it starts looking for the main node.The value of the main node is a file path that Angular is looking for - the main.ts under src/host folder.

Booking-UI:



EXPO UI:

A black screen with white text

Description automatically generated

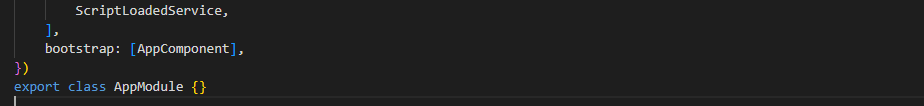
platformBrowserDynamic creates the platform. This function is also used to bootstrap the Angular Application. Here we specify the first module that should be loaded when the application starts for the first time - the root module.

Booking-UI:

A black screen with white text

Description automatically generated

EXPO UI:



where all the components/ directives/ pipes associated to this module is present and dependency to other module is also present – bootstrap.

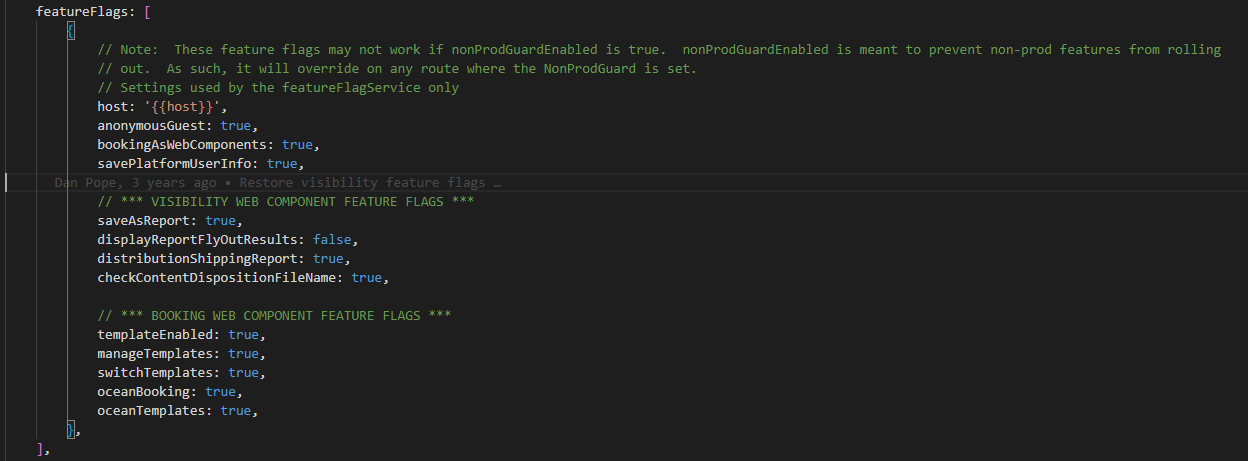
Where the starting component name has been specified - in this case the HostAppComponent.

By now Angular compiler has all the required information it needs to work.

**Micro frontend config:**

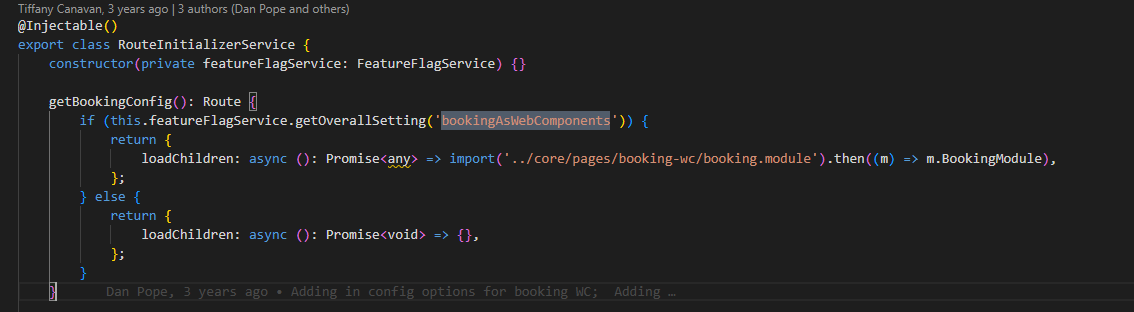
expo-ui\src\environments\environment.ts

set bookingAsWebComponents:true



service-endpoint-{env}.json

    "BookingUI": "http://localhost:8082/booking-ui-umd/versions.json"

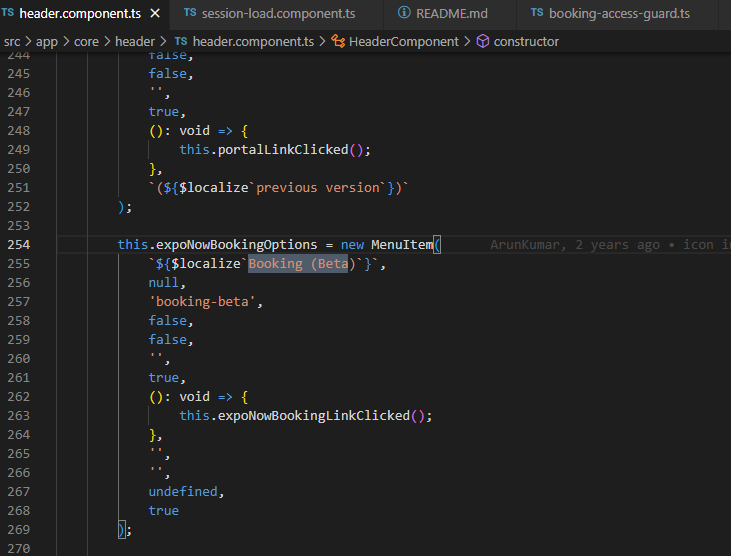


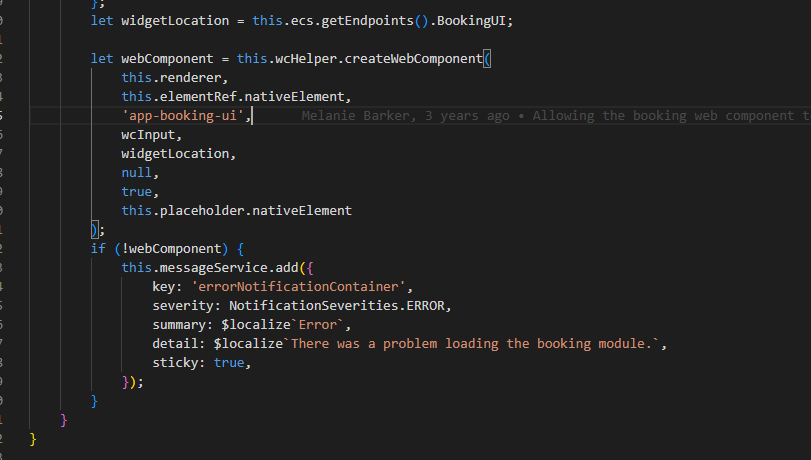
Load webcomponent from expo-ui.

RouteInitializerService

BookingModule

* + BookingWrapperComponent
    - this.wcHelper.createWebComponent





When building and loading as *\*a web component\**, you must build the booking module as a web component and make the web component available on a server or in your filesystem. Configuring the Angular skinny host with `jsLoad = true` will load the booking web component into the skinny host, instead of loading booking as an Angular module. To build and serve, use the commands:

A web component is a single javascript bundle that is loaded automatically by the browser.  This is commonly referred to as a 'umd' file.

To create this file, you basically concatenate a few of the normal build files together into a single javascript file.  Instead of doing this manually,

we chose to use ngx-build-plus which has a build option 'single-file'

Build the booking-ui as single-bundle using below command.

//Build

command = `node --max\_old\_space\_size=16384 node\_modules/@angular/cli/bin/ng build --deploy-url /assets/images/ --localize=true --extra-webpack-config extra-webpack.config.js --project ${args.projects} --single-bundle --sourceMap=false --configuration=${args.configuration}`;

    jsLoad: true, // 2020-06-23 - danp - Attempted to make these settings configurable on the UI but since it is used in the import

    //                 of host-app.module.ts, the UI cant change it since it is a part of the build process.

Additional:

main.js  
The JavaScript code that contains the application logic. Components, services and other utilities are getting compiled from typescript into this file.

polyfills.js  
All polyfills described on your polyfills.ts file are getting compiled here. A polyfill is a piece of code used to provide modern functionality on older browsers that do not natively support it.

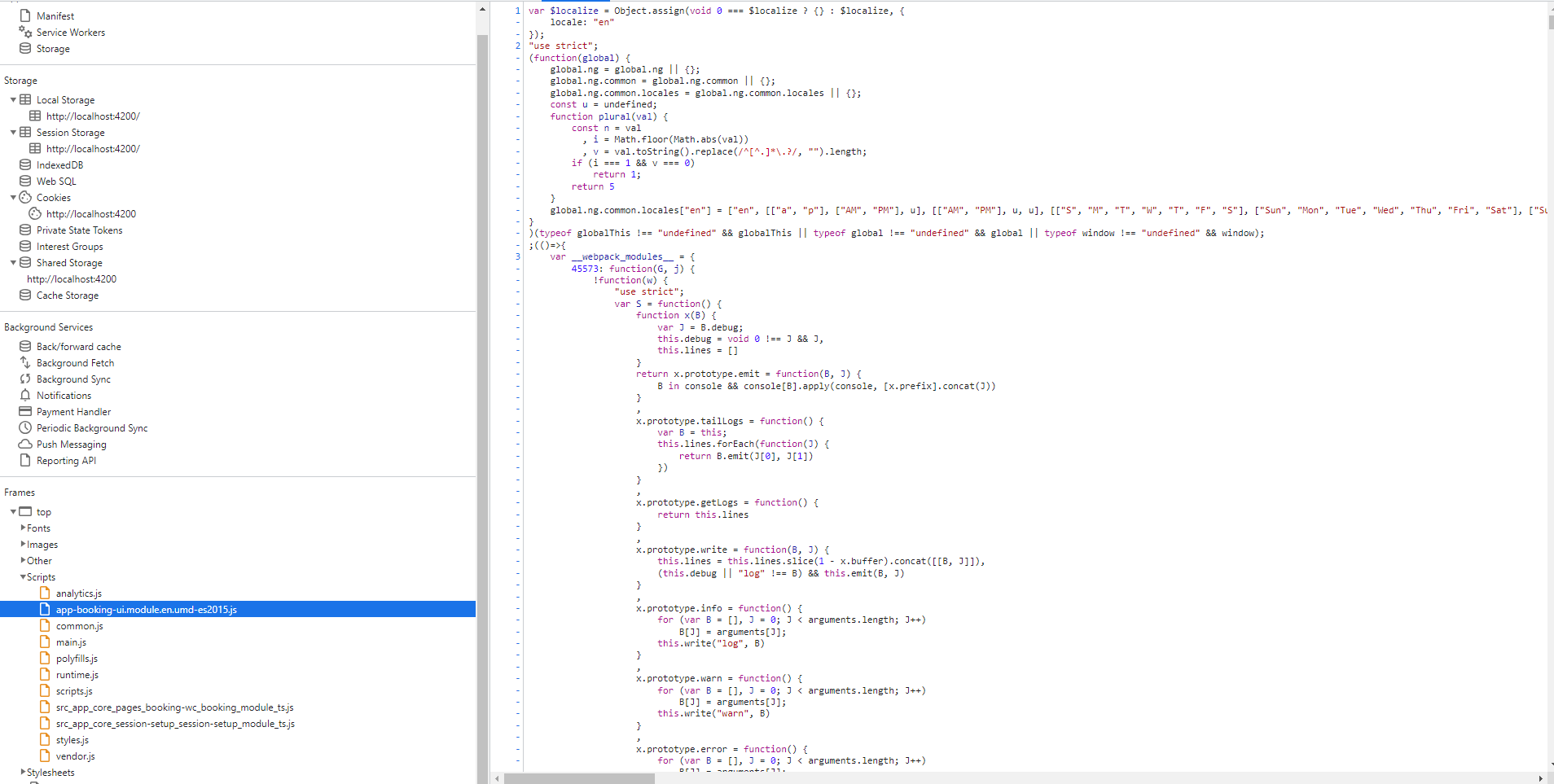
vendor.js  
A vendor bundle contains all project and framework related libraries that the application depends on. An application declares its dependencies first on the app.module.ts and then on other sub-modules across the app.

Booking-ui as angular application:

A screenshot of a computer

Description automatically generated

Booking-ui as web component with EXPO-UI:



<https://gitlab.expeditors.com/EnterpriseTechnology/booking-ui/-/blob/develop/README.md>

<https://gitlab.expeditors.com/EnterpriseTechnology/expo-ui/-/blob/develop/README.md>

<https://www.section.io/engineering-education/how-to-split-a-monolithic-architecture-into-micro-frontends/>

<https://medium.com/tenable-techblog/introducing-module-federation-c95e2551a5b9>