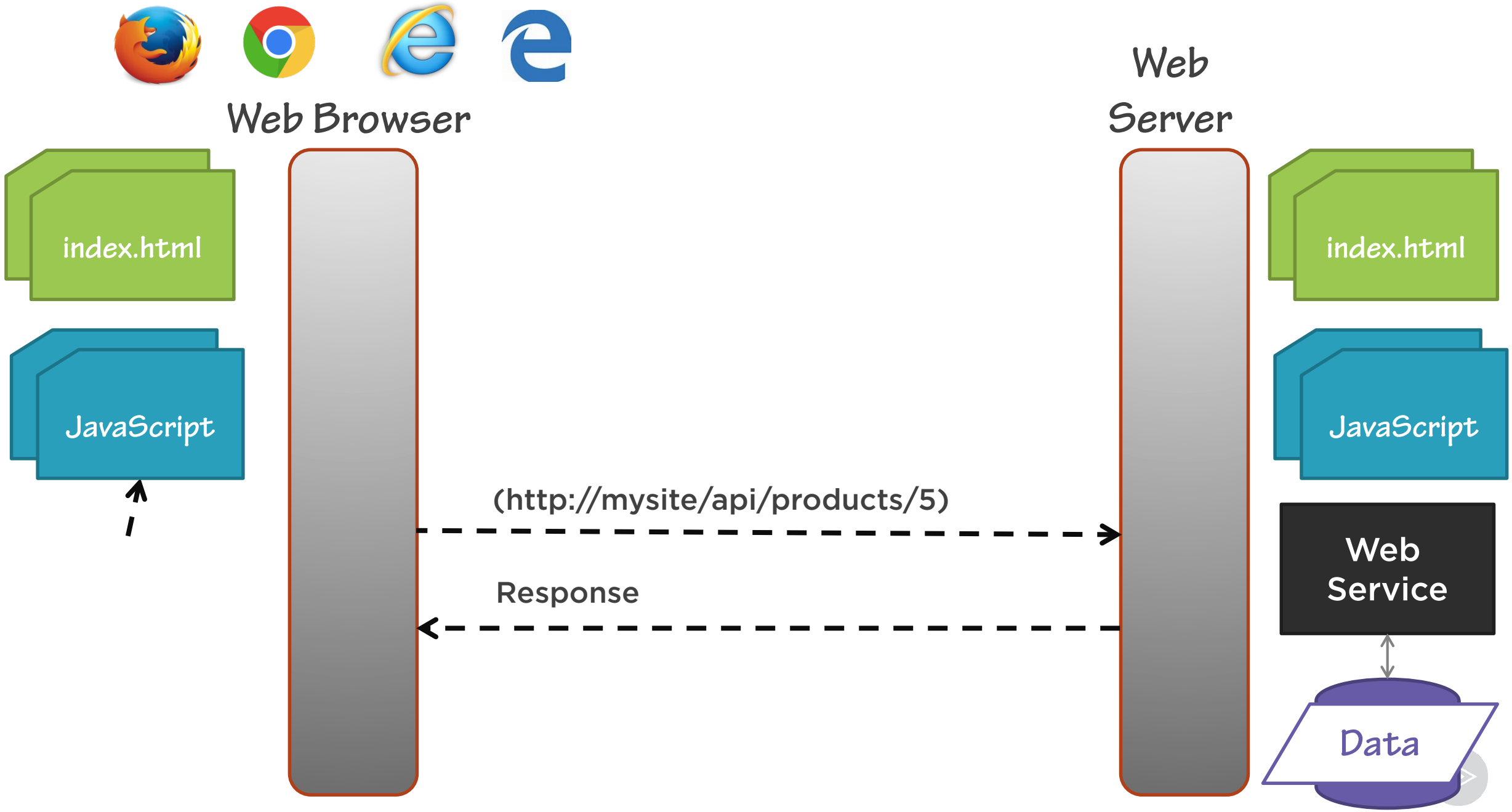
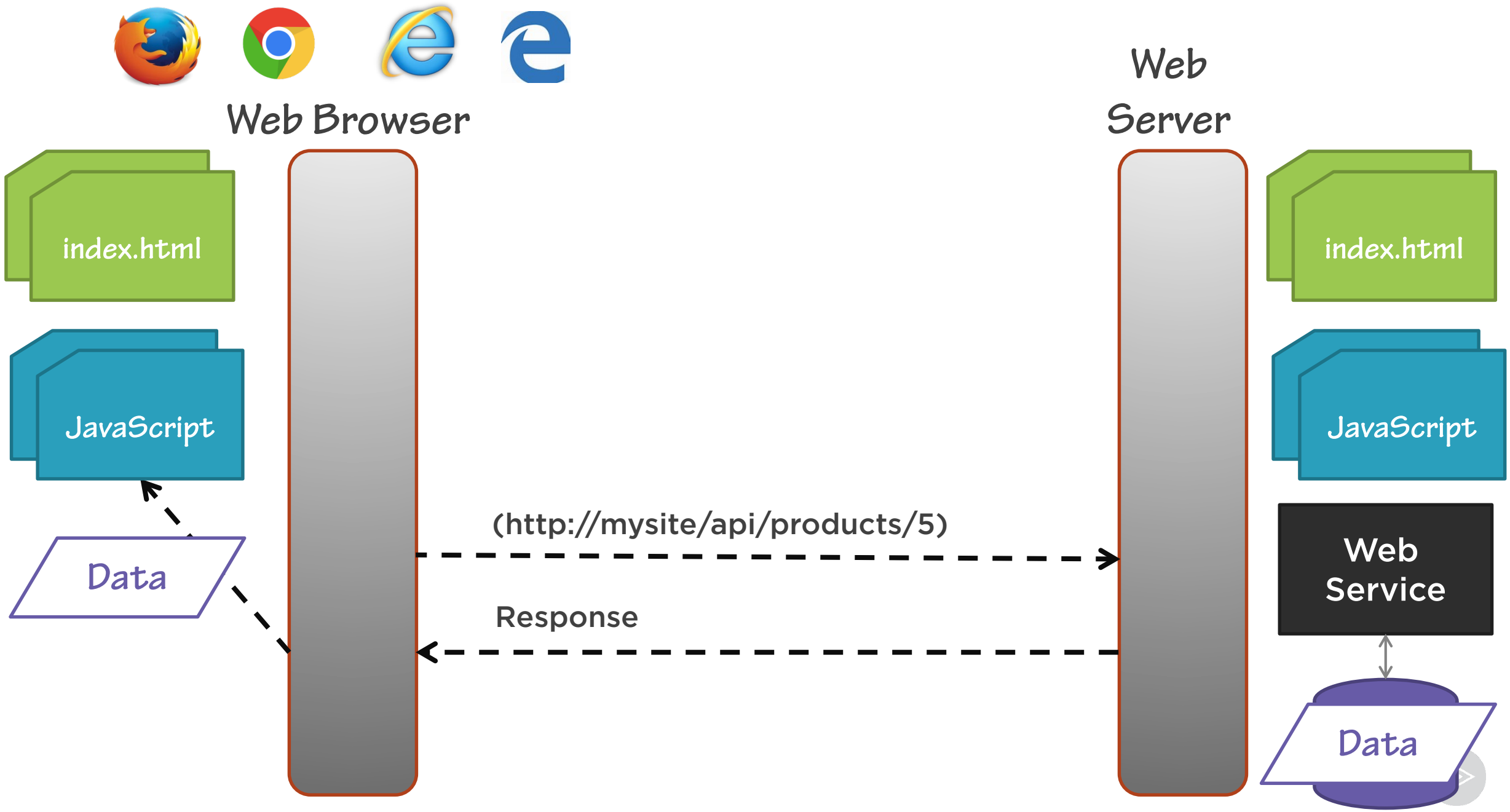
Create, Read, Update, Delete (CRUD)

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


Web Browser
Web Server
index.html
JavaScript
(http://mysite/api/products/5)
Response
index.html
JavaScript
Web Service
Data


Web Browser
Web Server
index.html
JavaScript
Data
(http://mysite/api/products/5)
Response
index.html
JavaScript
Web Service
Data

Module Overview

Data Access Service

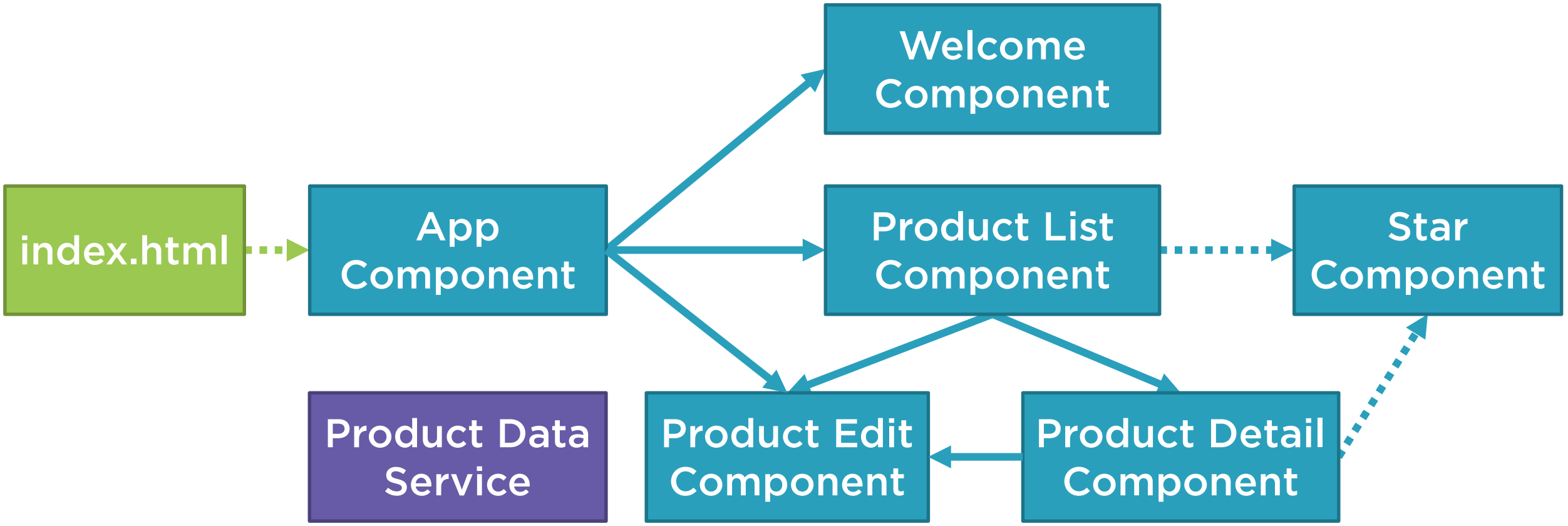
Creating Data

Reading Data

Updating Data

Deleting Data

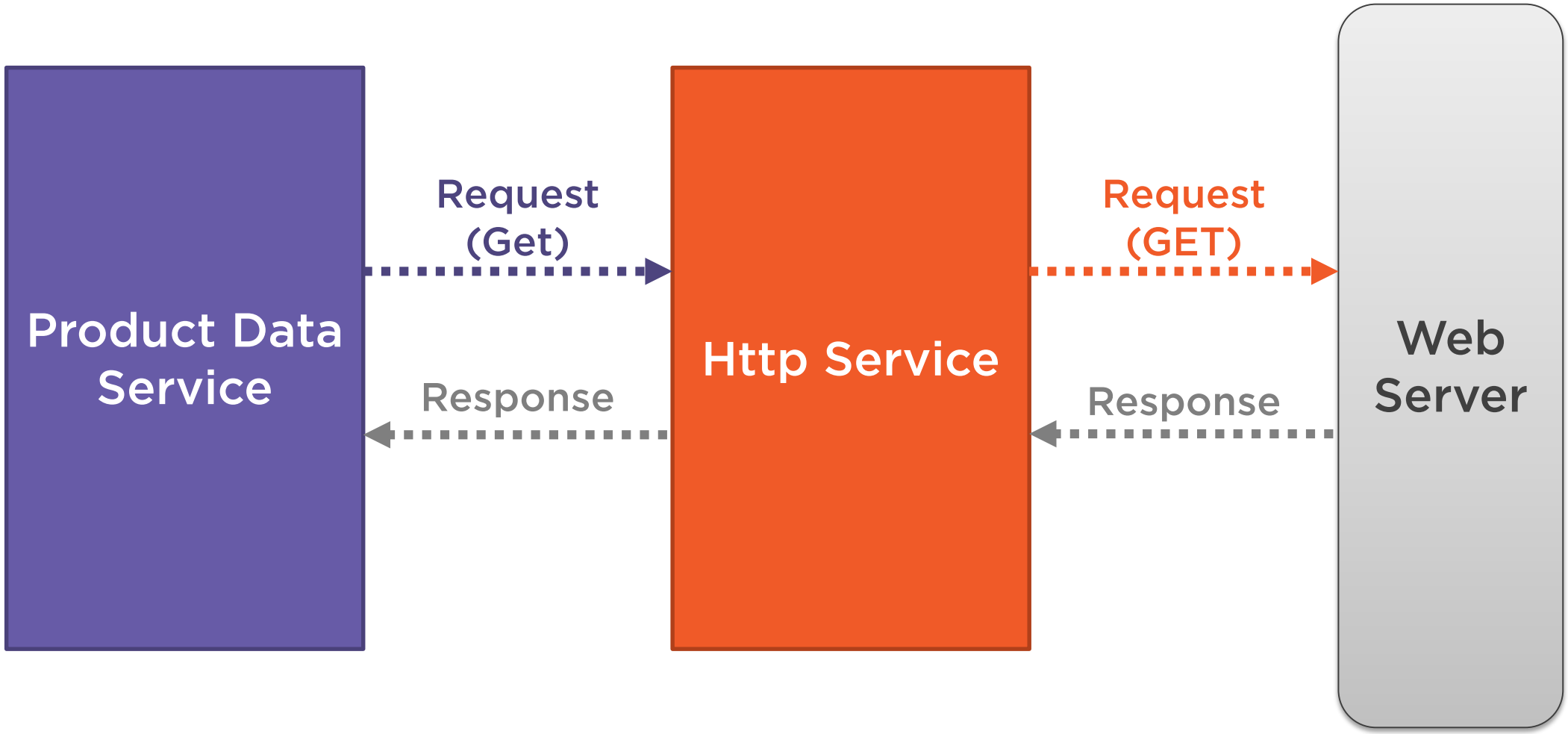
APM Sample Application Architecture



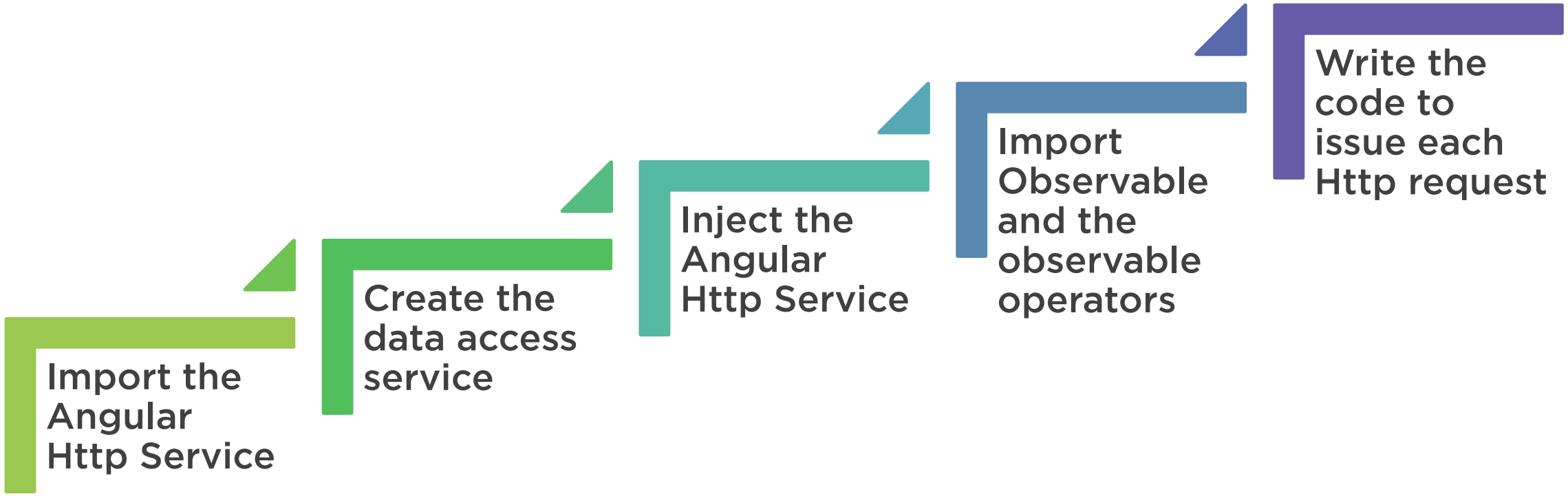
Why Build a Data Access Service?

- Separation of Concerns
- Reusability
- Data Sharing

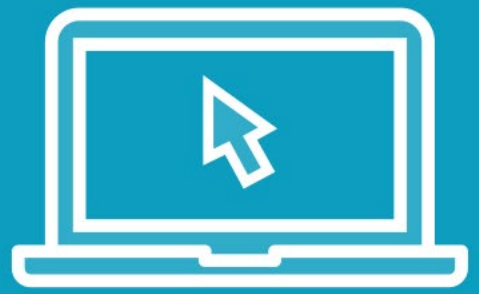
Sending an HTTP Request



Steps to Building a Data Access Service

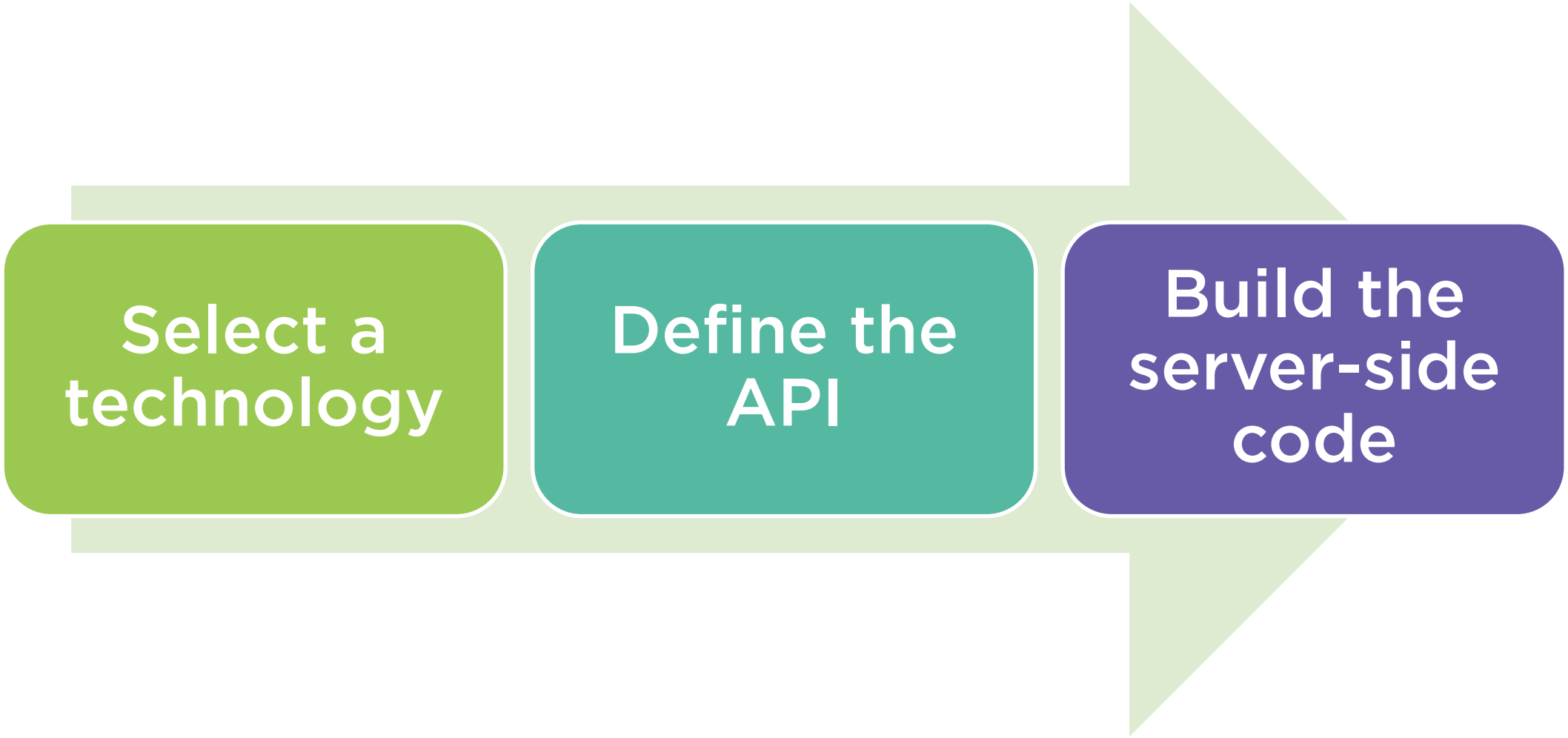


Demo



Building a Data Access Service

Setting up the Backend Server



Faking a Backend Server

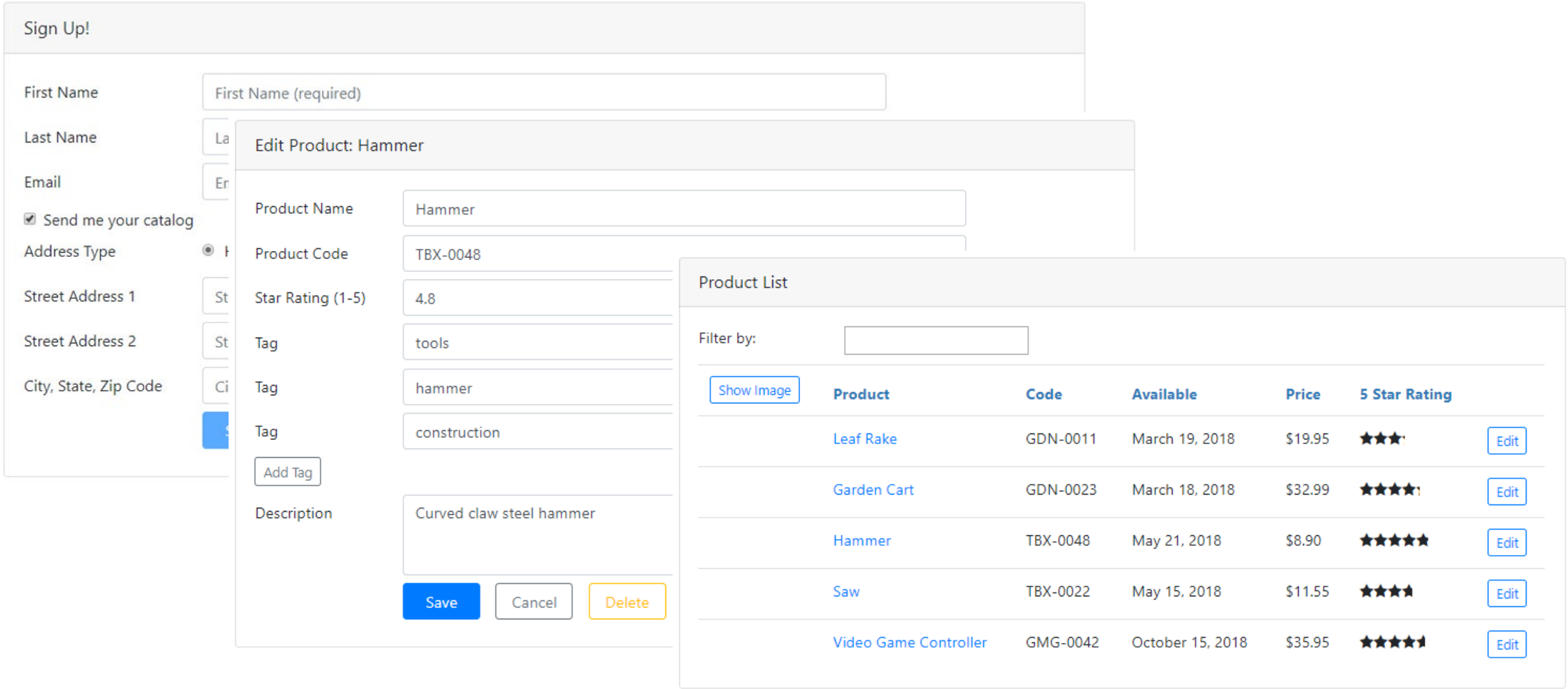
Directly return hard-coded data

Use a JSON file

Write our own code using MockBackend

Use angular-in-memory-web-api

Populating the Form with Data



HTTP Get Request

product.service.ts

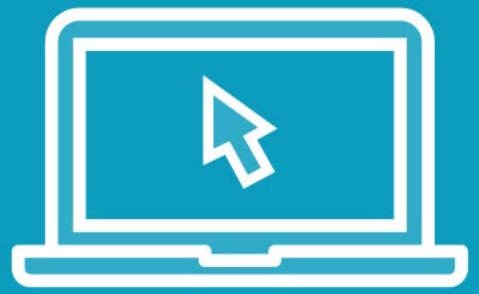
```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProduct(id: number): Observable<Product> {
     const url = `${this.baseUrl}/${id}`;
     return this.http.get<Product>(url);
  }
}
```

Calling the Data Access Service

product-edit.component.ts

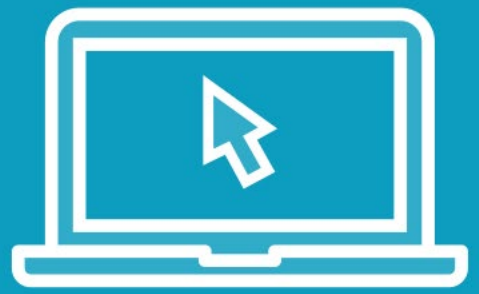
```
...

constructor(private productService: ProductService) { }
...

getProduct(id: number): void {
    this.productService.getProduct(id)
        .subscribe(
            (product: Product) => this.displayProduct(product),
            (error: any) => this.errorMessage = <any>error
        );
}
```

Demo



Populating the Form with Data: HTTP get

Demo



Populating the Form with Data: Subscribe

Saving Edits

Edit Product: Hammer

Product Name: Hammer

Product Code: TBX-0048

Star Rating (1-5): 4.8

Tag: tools — Delete Tag

Tag: hammer — Delete Tag

Tag: construction — Delete Tag

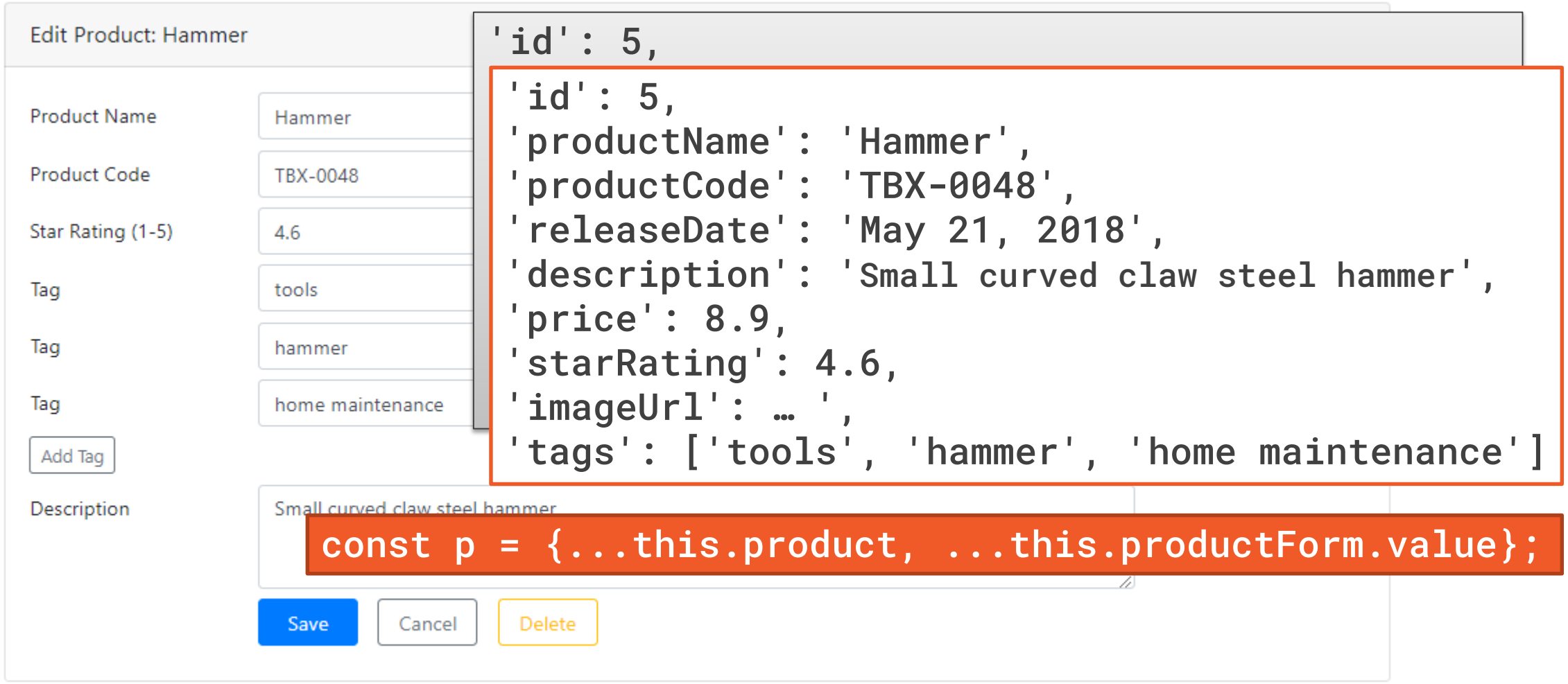
Add Tag

Description: Curved claw steel hammer

Save | Cancel | Delete

{ "productName": "Hammer", "productCode": "TBX-0048", "starRating": 4.8, "description": "Curved claw steel hammer", "tags": ["tools", "hammer", "construction"] }

Saving Edits



{ "productName": "Hammer", "productCode": "TBX-0048", "starRating": "4.6", "description": "Small curved claw steel hammer", "tags": ["tools", "hammer", "home maintenance"] }

Post vs Put

POST (api/products)

Posts data for a resource or set of resources

Used to:

- Create a new resource when the server assigns the Id
- Update a set of resources

Not idempotent

PUT (api/products/5)

Puts data for a specific resource with an Id

Used to:

- Create a new resource when the client assigns the Id
- Update the resource with the Id

Idempotent

HTTP Put Request

product.service.ts

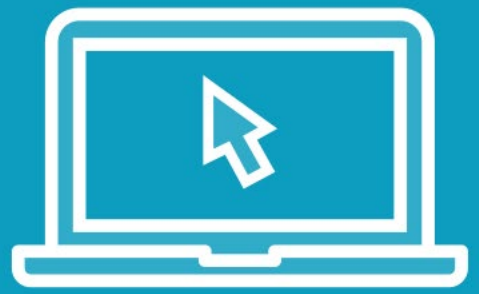
```
...
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  updateProduct(product: Product): Observable<Product> {
    const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

    const url = `${this.baseUrl}/${product.id}`;
    return this.http.put<Product>(url, product, { headers: headers });
  }
}
```

Calling the Data Access Service

product-edit.component.ts

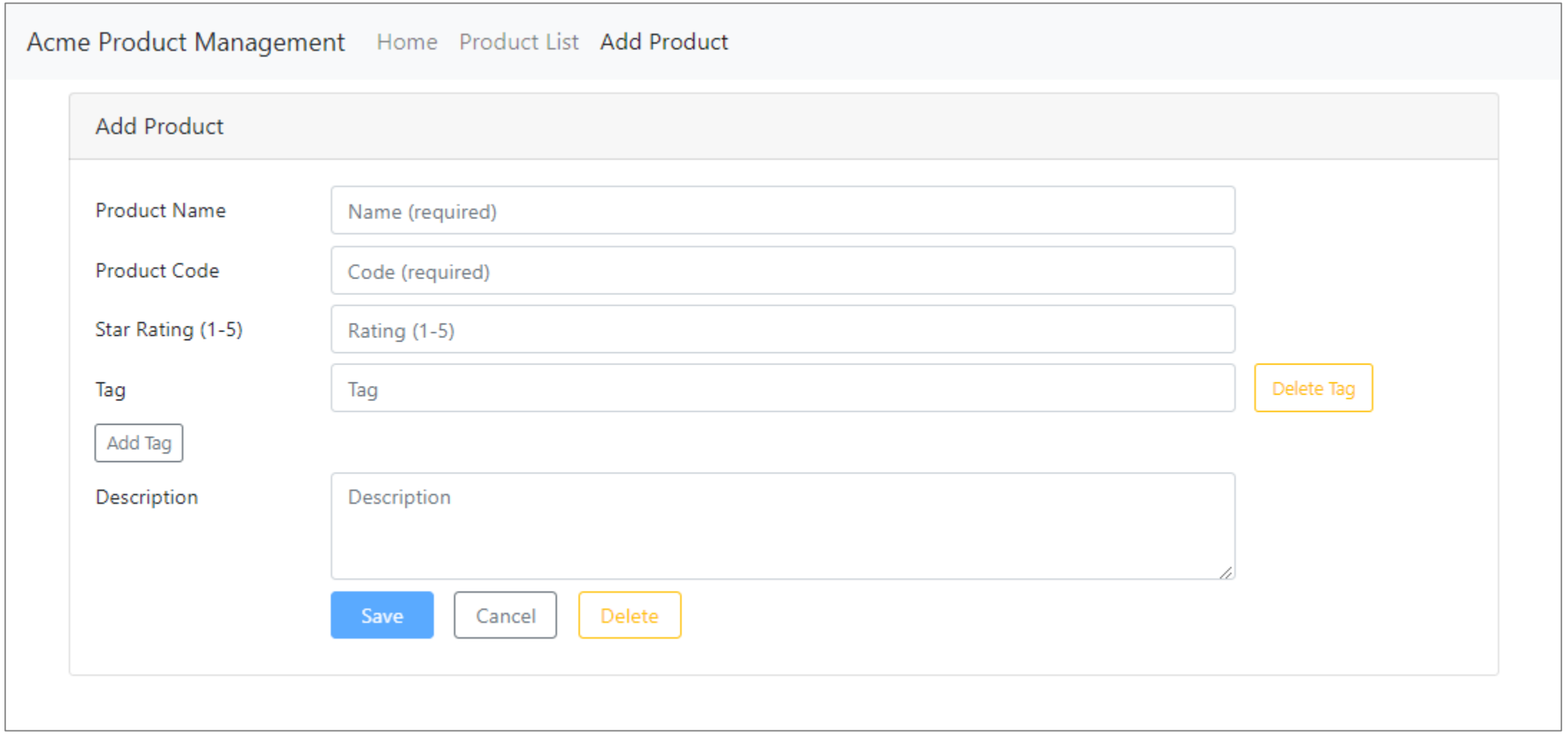
```
editProduct: void {
    this.productService.updateProduct(p)
        .subscribe(
            () => this.onSaveComplete(),
            (error: any) => this.errorMessage = <any>error
        );
}
```

Demo



Saving Edits

Creating New Items



Initializing an Object

product.service.ts

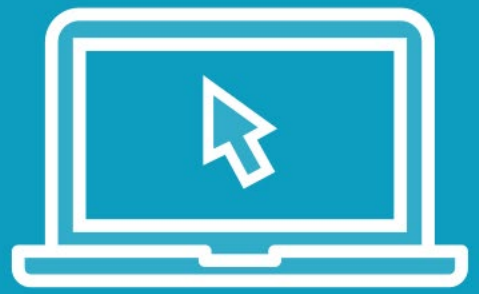
```
initializeProduct(): Product {
  return {
      id: 0,
      productName: null,
      productCode: null,
      tags: [''],
      releaseDate: null,
      price: null,
      description: null,
      starRating: null,
      imageUrl: null
  }
}
```

HTTP Post Request

product.service.ts

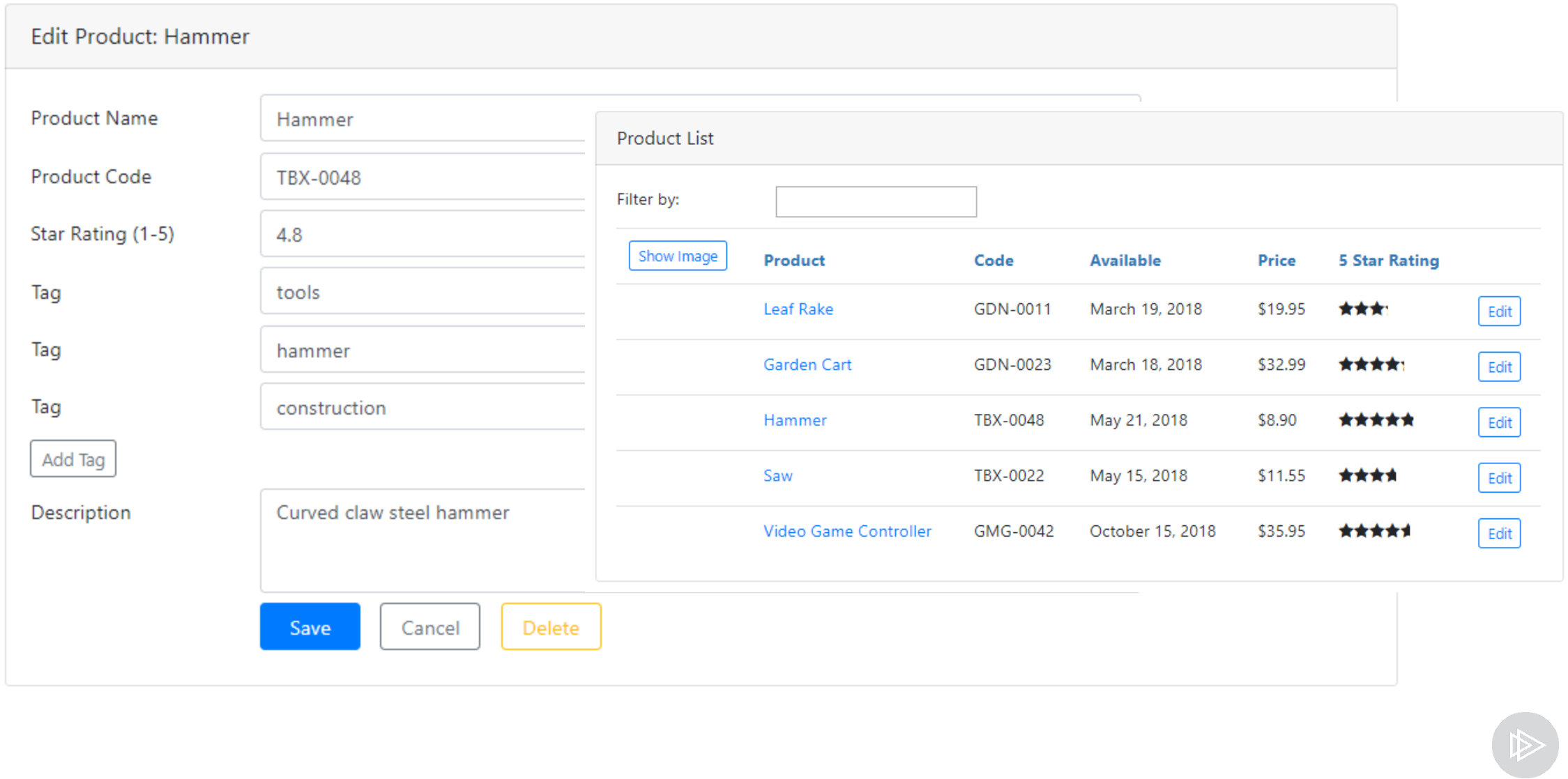
```
...
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  createProduct(product: Product): Observable<Product> {
   const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

   return this.http.post<Product>(this.baseUrl, product,
                                  { headers: headers });
  }
}
```

Demo



Creating New Items

Deleting an Existing Item



HTTP Delete Request

product.service.ts

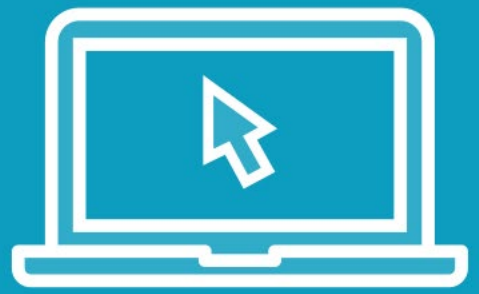
```
...
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  deleteProduct(id: number): Observable<{}> {
   const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

   const url = `${this.baseUrl}/${id}`;
   return this.http.delete<Product>(url, { headers: headers });
  }
}
```

Calling the Data Access Service

product-edit.component.ts

```
deleteProduct: void {
    this.productService.deleteProduct(this.product.id)
        .subscribe(
            () => this.onSaveComplete(),
            (error: any) => this.errorMessage = <any>error
        );
}
```

Demo



Deleting an Existing Item

CRUD Checklist: Import the Http Service

app.module.ts

```
...
import { HttpClientModule }
    from '@angular/common/http';

@NgModule({
  imports: [ HttpClientModule ],
  ...
})
export class AppModule { }
```

Add HttpClientModule to the imports array of an Angular Module

CRUD Checklist: Data Access Service

Import what we need

product.service.ts

```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';
```

CRUD Checklist: Data Access Service

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

product.service.ts

```
...
export class ProductService {

  constructor(private http: HttpClient) { } }
```

CRUD Checklist: Data Access Service

product.service.ts

```
...

getProduct ...
createProduct ...
updateProduct ...
deleteProduct ...
```

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

CRUD Checklist: Data Access Service

product.service.ts

```
...

const url =
  `${this.baseUrl}/${id}`;
return this.http.get(url);
```

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

CRUD Checklist: Data Access Service

product.service.ts

```
...
const url =
 `${this.baseUrl}/${id}`;

return this.http.get<Product>(url)
 .pipe(
   .catchError(this.handleError)
 );
```

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

Add error handling

CRUD Checklist: Using the Service

product-edit.component.ts

```
...
constructor(private ps: ProductService) { }
```

Inject the Data Access Service

CRUD Checklist: Using the Service

product-edit.component.ts

```
...

this.ps.getProduct(id)
  .subscribe();
```

Inject the Data Access Service

Call the subscribe method of the returned observable

CRUD Checklist: Using the Service

product-edit.component.ts

```
...

this.ps.getProduct(id)
 .subscribe(
  (product: Product) =>
    this.onRetrieved(product)
);
```

Inject the Data Access Service

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

CRUD Checklist: Using the Service

product-edit.component.ts

```
...

this.ps.getProduct(id)
 .subscribe(
  (product: Product) =>
    this.onRetrieved(product),
  (error: any) =>
    this.errorMessage = error
);
```

Inject the Data Access Service

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

Provide an error function to handle any returned errors

Summary

Data Access Service

Creating Data

Reading Data

Updating Data

Deleting Data