Using Data from an API

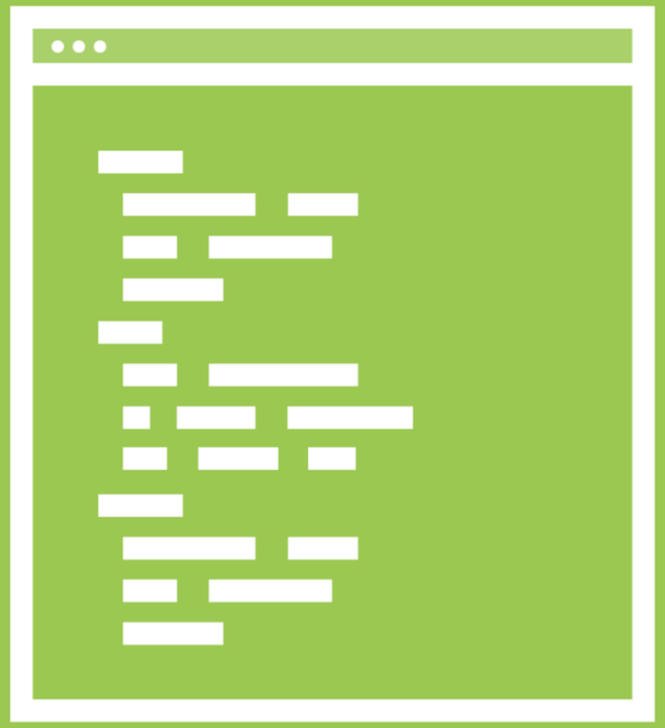
Gill Cleeren
CTO Xpirit Belgium

@gillcleeren – xpirit.com/gill

Module overview

- Working with an API
- Accessing API data using HttpClient
- Managing the application state
- Storing data locally

Working with an API



Hard coded data

Almost every app will use “real” data

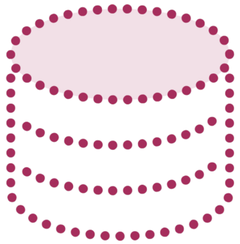
Working with Data

Entity Framework Core

API

Local Storage

Creating an API

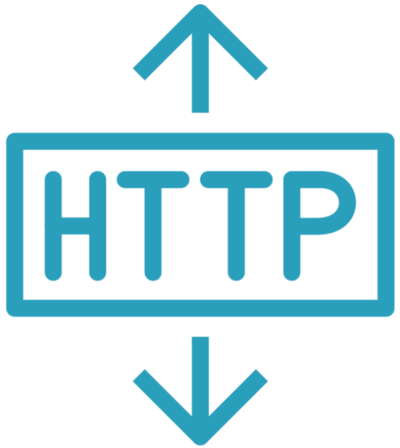


Uses “just” the data

JSON or XML

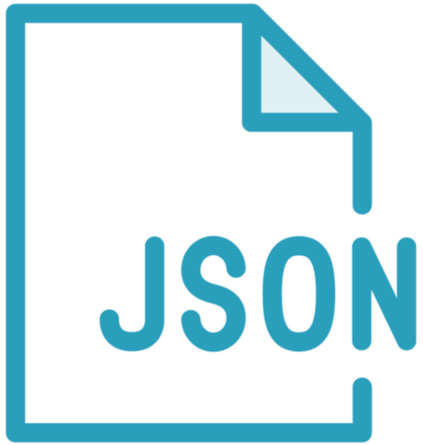
Open for many types of clients

Creating a RESTful API



HTTP request GET, POST, PUT...

Resources with URLs



Responses in JSON

Status codes 200, 404...

Accessing a REST API

GET /api/employee

200 + json response

API

HTTP Verbs

GET

POST

PUT

DELETE

JSON Response

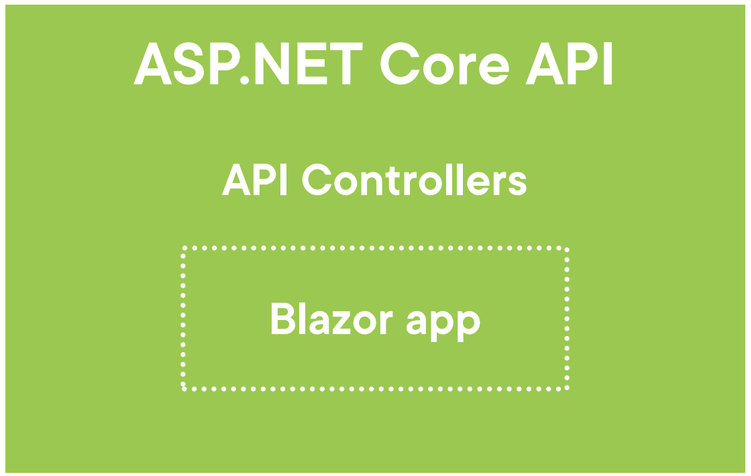

```
[
    {
        "employeeId": 1,
        "firstName": "Bethany",
        "lastName": "Smith",
        "birthDate": "1979-01-16T00:00:00",
        "email": "bethany@bethanyspieshop.com",
        "street": "Grote Markt 1",
        "zip": "1000",
        "city": "Brussels",
        "countryId": 1,
        "country": null,
        "phoneNumber": "324777888773",
        "smoker": false,
        "maritalStatus": 1,
        "gender": 1,
        "comment": "Lorem Ipsum",
        "joinedDate": "2015-03-01T00:00:00",
        "exitDate": null,
        "jobCategoryId": 1,
        "jobCategory": null,
        "latitude": 50.8503,
        "longitude": 4.3517,
        "imageContent": null,
        "imageName": null
    },
    {
        "employeeId": 2,
        "firstName": "aa",
        "lastName": "aa",
        "birthDate": "2022-07-24T10:29:07.143",
        "email": "aa",
        "street": "aa",
        "zip": "a",
        "city": "aa",
        "countryId": 1,
        "country": null,
        "phoneNumber": "",
        "smoker": true,
        "maritalStatus": 0,
        "gender": 0,
        "comment": null,
        "joinedDate": "2022-07-24T10:29:07.161",
        "exitDate": null,
        "jobCategoryId": 3,
        "jobCategory": null,
        "latitude": 2,
        "longitude": 1,
        "imageContent": null,
        "imageName": null
    },
```

Demo

Exploring the API

- ASP.NET Core REST API

Sidestep: ASP.NET Core Hosted



ASP.NET Core project exposes the API project
and hosts the Blazor app too.

Demo

Moving Blazor to ASP.NET Core Hosted

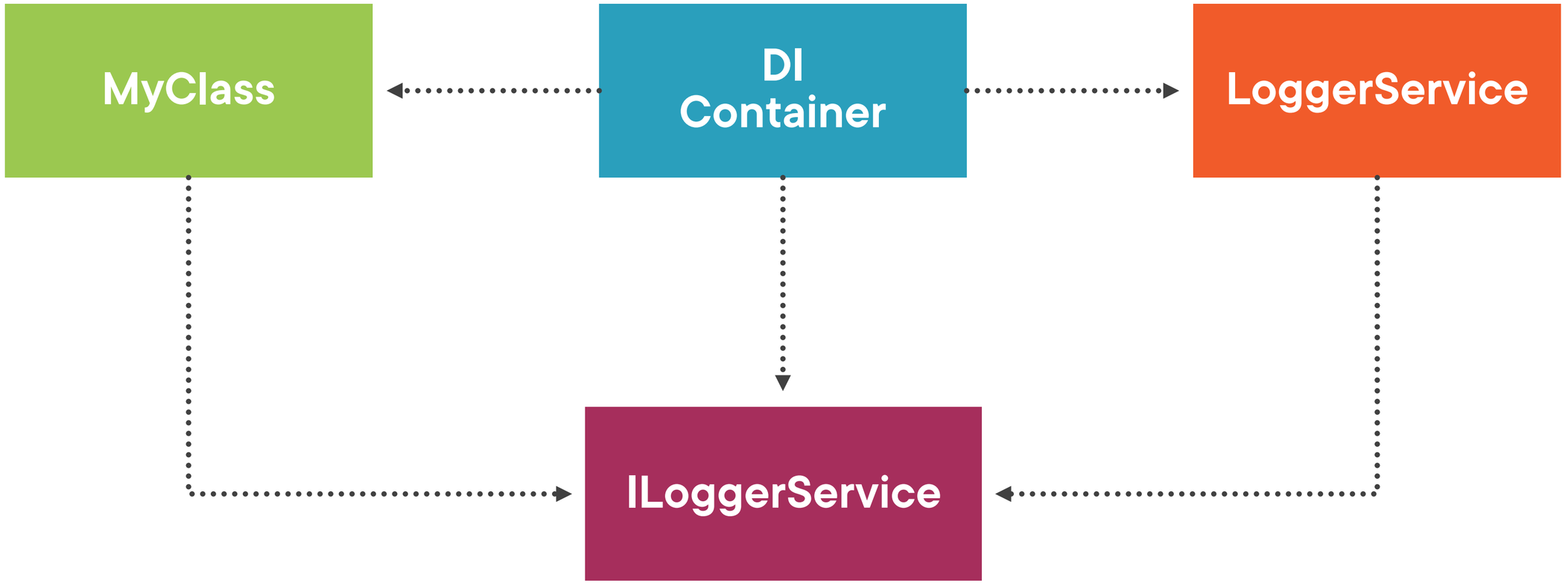
Accessing API Data Using HttpClient

Interacting with the REST API

HttpClient

IHttpClientFactory

Sidestep: Dependency Injection (DI)



```
builder.Services.AddScoped(sp =>
    new HttpClient
    {
        BaseAddress = new Uri("http://<your-api-endpoint>")
    }
);
```

Registering the HttpClient

```
[Inject]
public HttpClient HttpClient { get; set; }
```

Accessing the HttpClient in a Component

```
protected override async Task OnInitializedAsync()
{
    Employees = await HttpClient.GetFromJsonAsync<Employee[]>("api/employee");
}
```

Working with the JSON Helper Methods

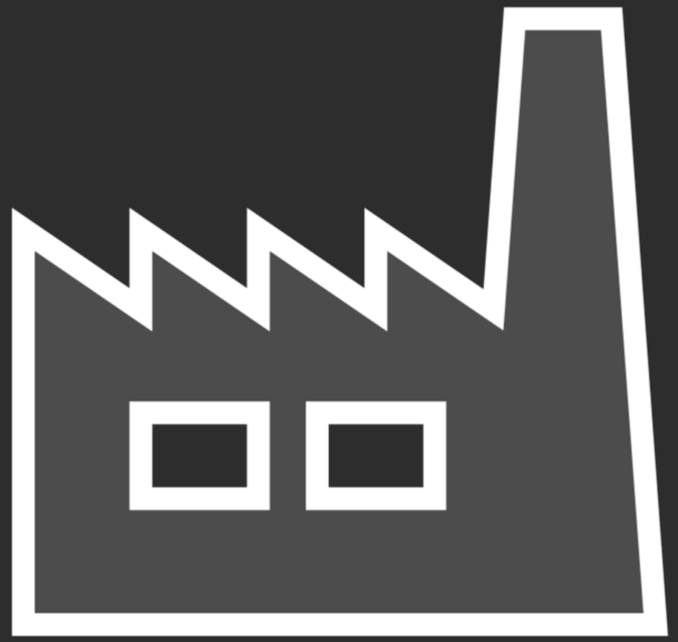
Available Methods

GetFromJsonAsync()

PostAsJsonAsync()

PutAsJsonAsync()

DeleteAsync()



IHttpClientFactory

Used to configure and create HttpClient instances in a central location

Support for named and typed HttpClient

Requires Microsoft.Extensions.Http package

```
builder.Services.AddHttpClient
  <IEmployeeDataService, EmployeeDataService>
    (client => client.BaseAddress = new Uri("https://localhost:44340/"));
```

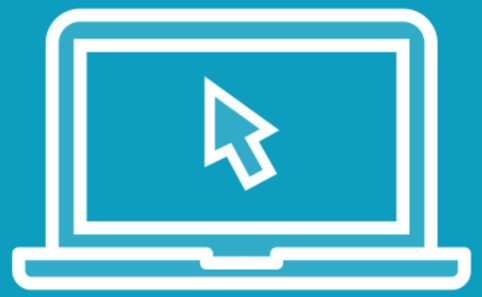
Registering in the Program

Typed client is used here

Creating a Service Class

```
public class EmployeeDataService : IEmployeeDataService
{
    private readonly HttpClient _httpClient;

    public EmployeeDataService(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }
}
```

Demo

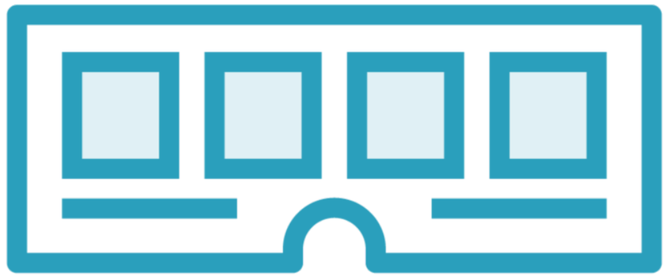


Registering the HttpClient

Creating a data service

Updating the pages to use data from the API

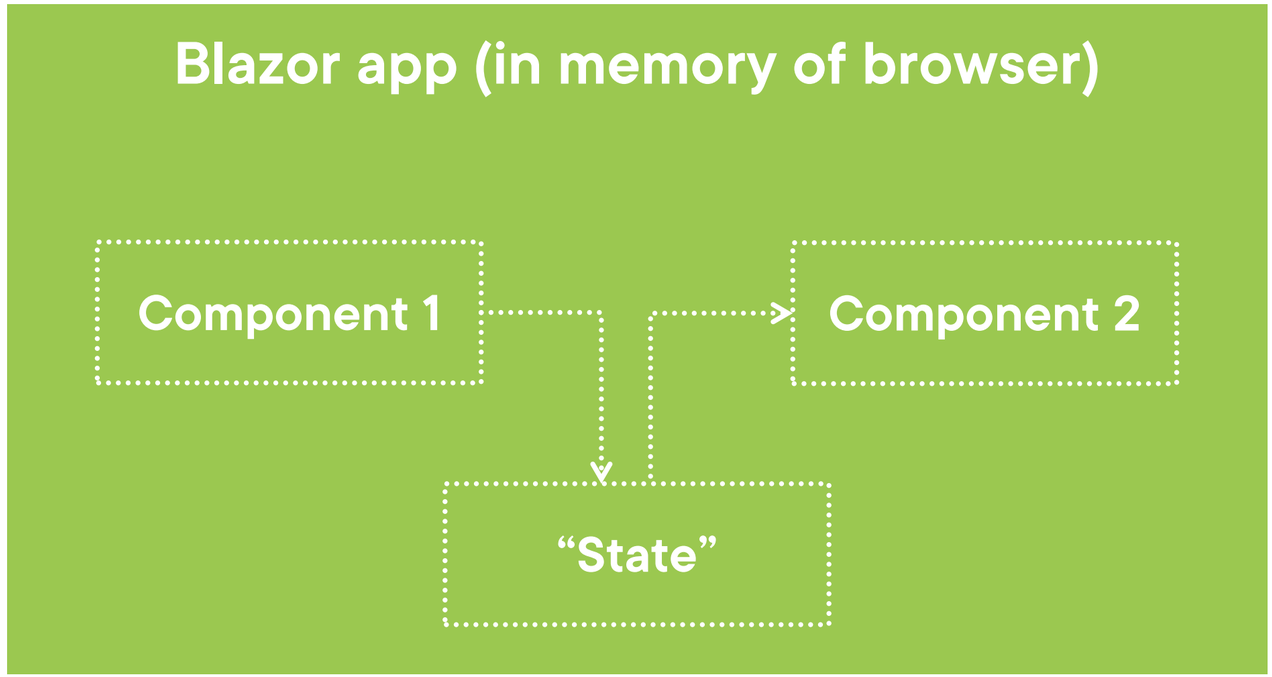
Managing the Application State



Application state

- Blazor WASM is an application that's running in the memory of the browser
- By default, each component is an island that's recreated every time

Application State



```
public class ApplicationState
{
    public int NumberOfMessages { get; set; } = 0;
}
```

Creating an Application State Class

```
builder.Services.AddScoped<ApplicationState>();
```

Adding an Instance to the DI Container

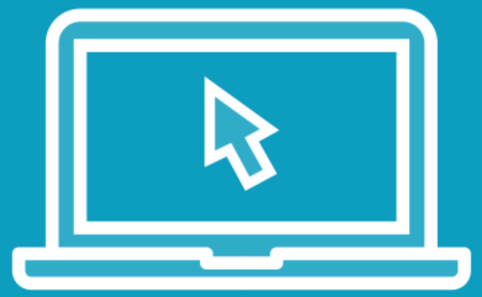
```
[Inject]
public ApplicationState? ApplicationState { get; set; }

int a = ApplicationState.NumberOfMessages;
```

Accessing the Application State from Components

Application State

This type of state is in-memory and will be removed when the application restarts!

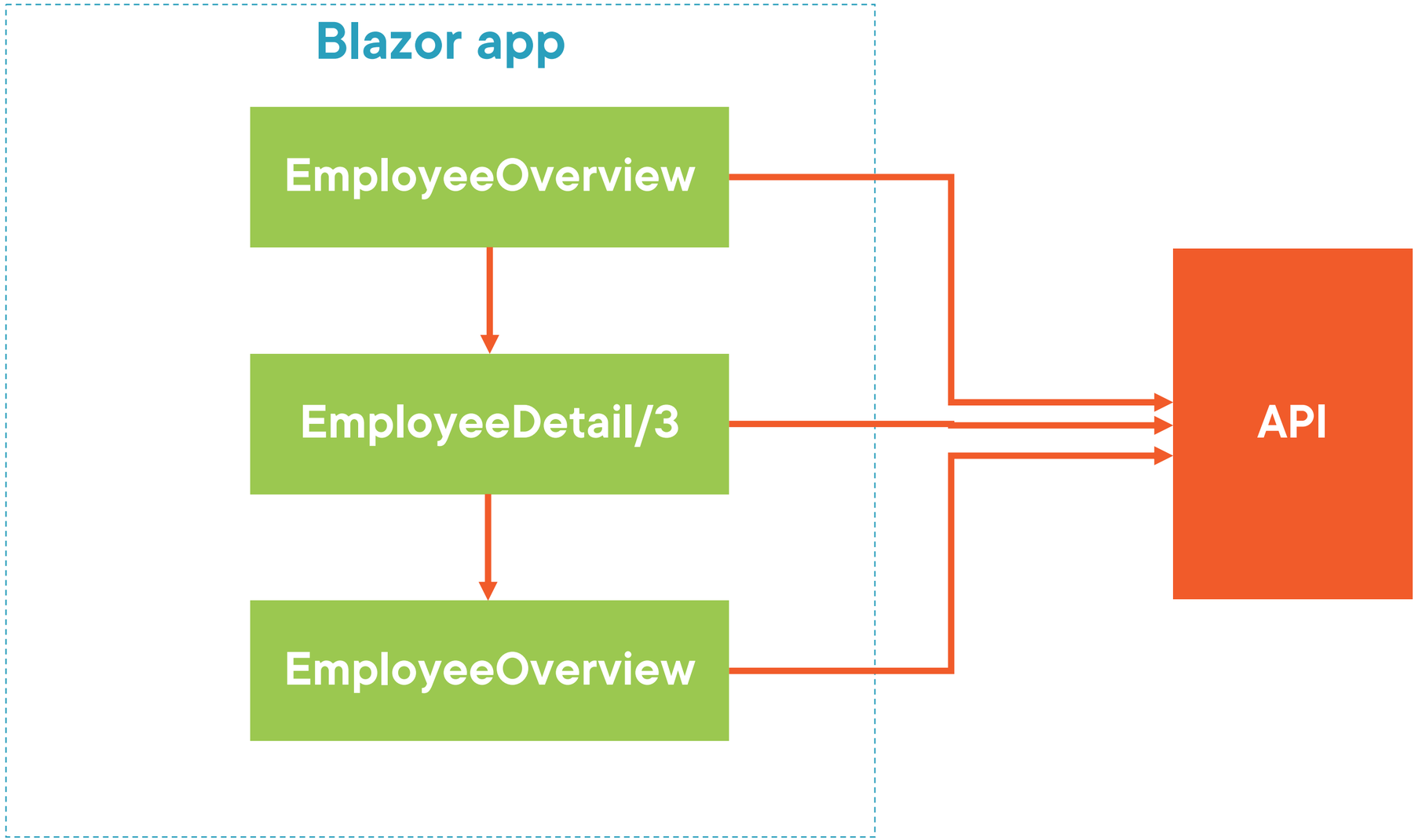
Demo



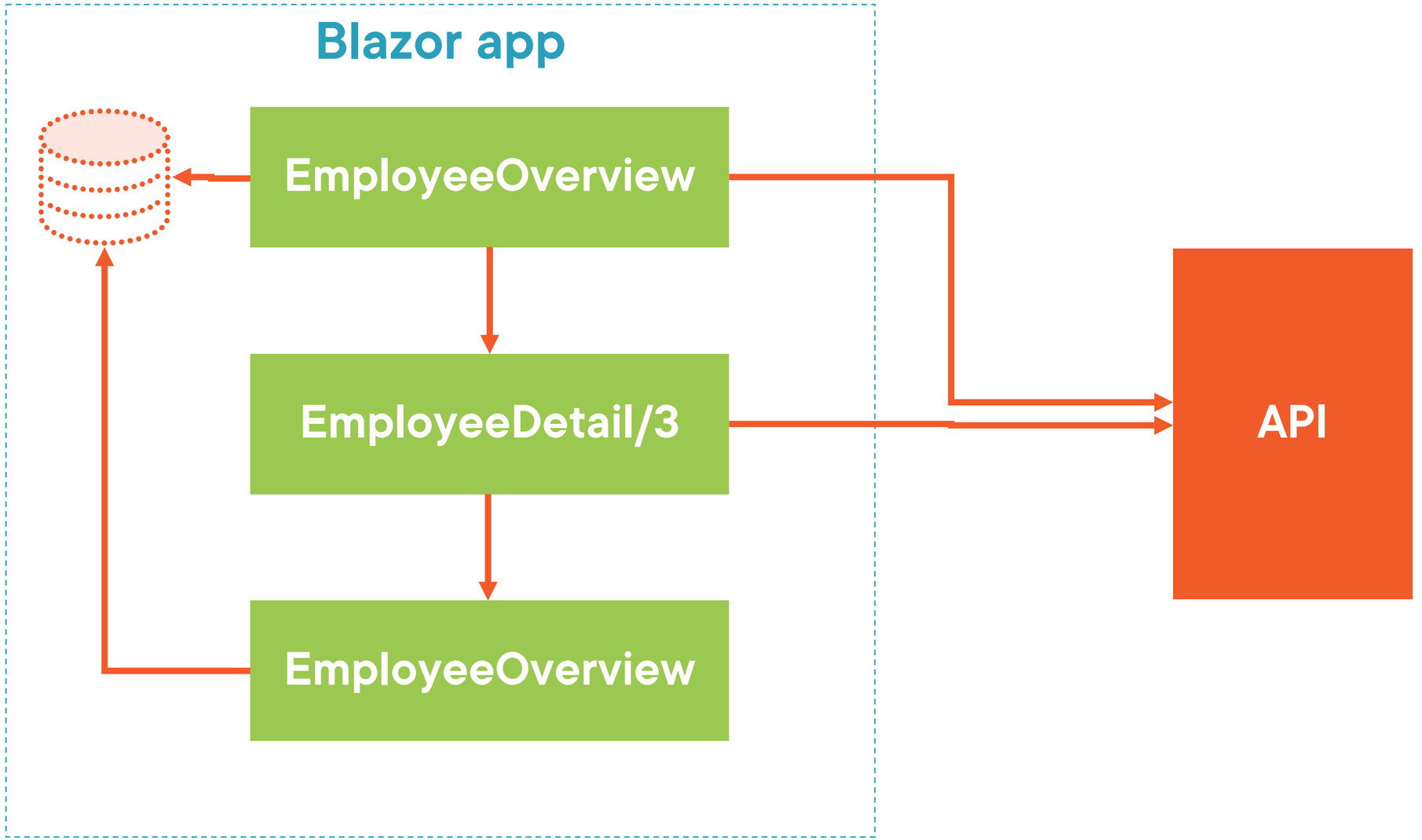
Adding application state

Storing Data Locally

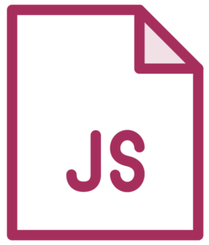
Unneeded API Calls



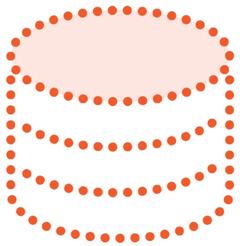
Adding Local Storage



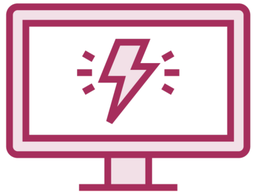
Storing Data Locally



Made possible through the browser, accessible using JavaScript

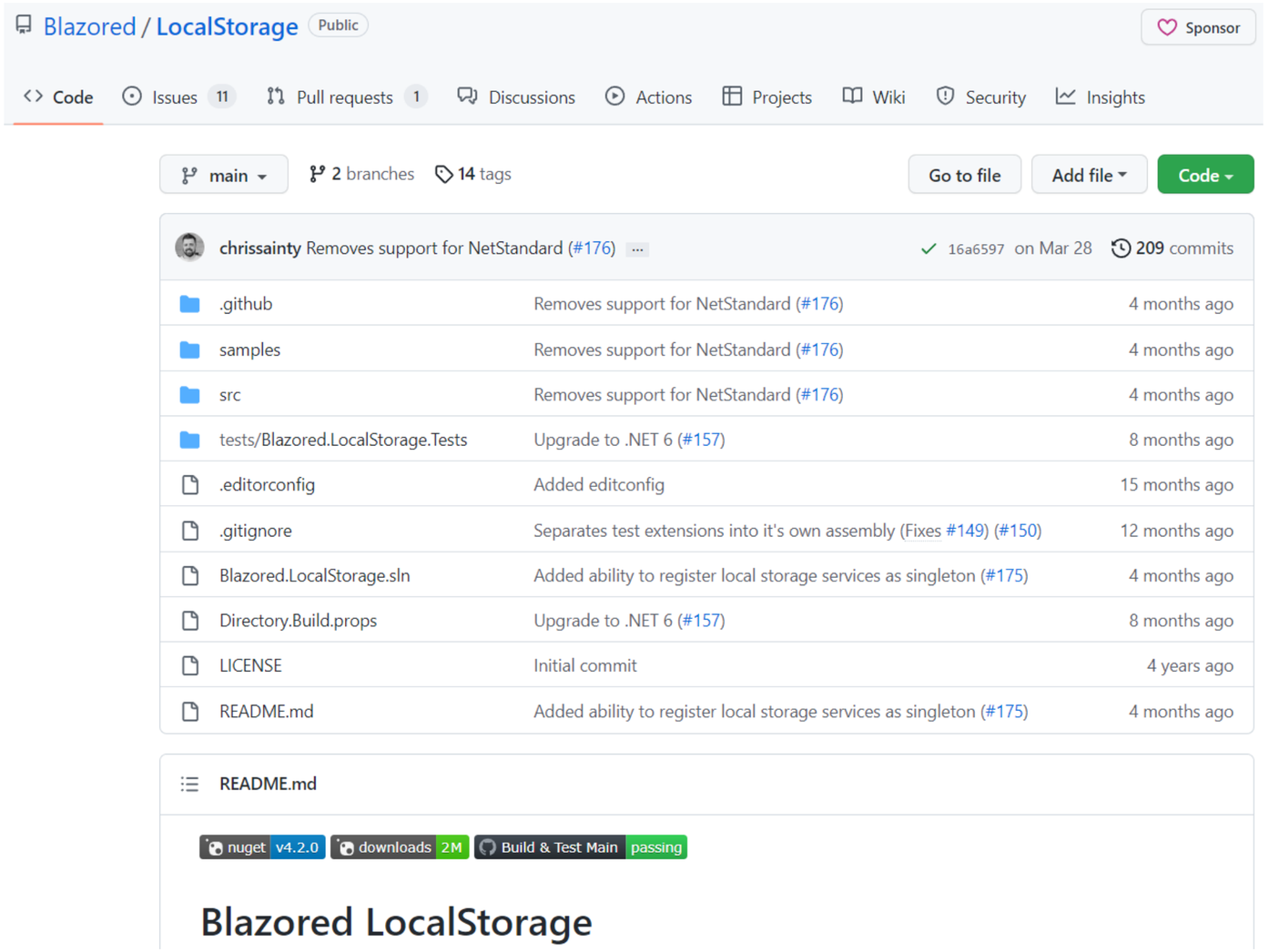


SessionStorage or LocalStorage



Possible to use from Blazor WASM too

Using Blazored LocalStorage



```
@inject Blazored.LocalStorage.ILocalStorageService localStorage

var firstName = await localStorage.GetItemAsync<string>("EmployeeFirstName");
```

Using ILocalStorageService

Available APIs

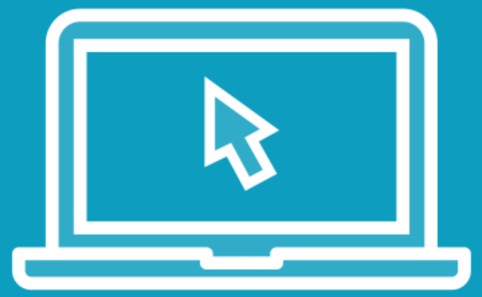
SetItem()
SetItemAsync()

GetItem()
GetItemAsync()

ContainKey()
ContainKeyAsync()

RemoveItem()
RemoveItemAsync()

Demo



Adding the Blazored.LocalStorage package

Extending the service with local storage support

Summary

- APIs offer us a way to access remote data
- Use HttpClient and IHttpClientFactory to access remote API
- Storing data locally will reduce load on API

Up next:
Creating a form