Umbraco 14 Workshop

Code of conduct

In our meetings we ask everyone to:

- **Be aware of others**
- **Be friendly and patient**
- **Be welcoming and respectful**
- **Be open to all questions and viewpoints**
- **Be understanding of differences**
- **Be kind and considerate**

Wifi Details

Umbraco Guest
OwnTheExperience!

System Requirements

- DotNet 8
- Node 20
- Visual Studio Code
  - C# Dev Kit

Workshop Agenda

- Getting started
  - Installing Umbraco 14
  - Creating our first local extension

  https://docs.umbraco.com/umbraco-cms/tutorials/creating-your-first-extension

- Kevin Jumps blog series
  - Setting up
  - Entry points
  - Communicating with the server

  https://dev.to/kevinjump/series

You can find all the resource links here

https://github.com/whitter/umbraco-14-workshop

The purpose of this workshop

To get you started

How to install Umbraco 14

- Install the Umbraco template

```
dotnet new install Umbraco.Templates
```

- Create an Umbraco project

```
dotnet new umbraco -n MyProject
```

Then dotnet run etc

https://docs.umbraco.com/umbraco-cms/fundamentals/setup/install

What technologies are used?

- Web components
- Lit (Google)
- RxJS

  A library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code.

- Typescript
- Vite
- Storybook

Creating an extension with Vite, Typescript & Lit

- Create your App_Plugins folder in Umbraco root
- In the App_Plugins folder run

```
npm create vite@latest my-typescript-extension -- --template lit-ts
```

- Then install

```
npm install
```

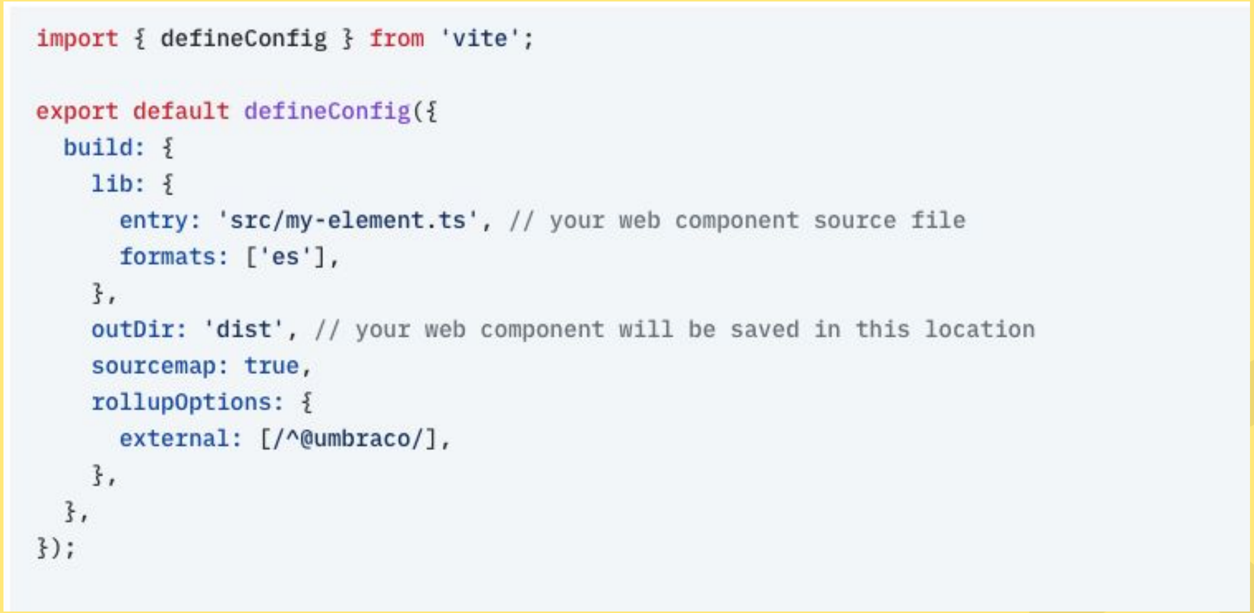
- Then install beta back office

```
npm install -D @umbraco-cms/backoffice
```

https://docs.umbraco.com/umbraco-cms/tutorials/creating-your-first-extension

Creating an extension with Vite, Typescript & Lit

- Create a vite.config.ts in /App_Plugins/my-typescript-extension/



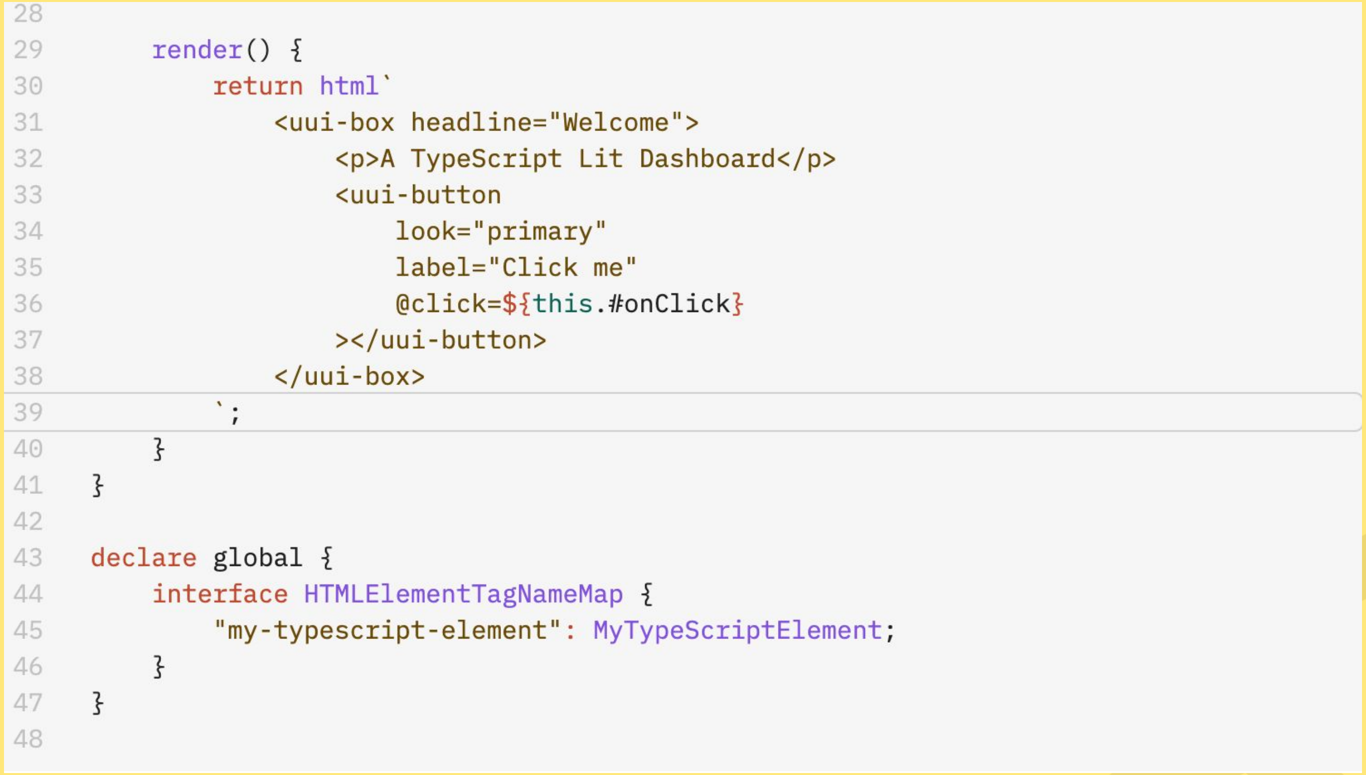
```ts
import { defineConfig } from 'vite';

export default defineConfig({
  build: {
    lib: {
      entry: 'src/my-element.ts', // your web component source file
      formats: ['es'],
    },
    outDir: 'dist', // your web component will be saved in this location
    sourcemap: true,
    rollupOptions: {
      external: [/^@umbraco/],
    },
  },
});
```

https://docs.umbraco.com/umbraco-cms/v/14.latest-rc/tutorials/creating-your-first-extension

Update /App_Plugins/my-typescript-extension/src/my-element.ts

```typescript
import {
    LitElement,
    html,
    customElement,
} from "@umbraco-cms/backoffice/external/lit";
import { UmbElementMixin } from "@umbraco-cms/backoffice/element-api";
import {
    UmbNotificationContext,
    UMB_NOTIFICATION_CONTEXT,
} from "@umbraco-cms/backoffice/notification";

@customElement("my-typescript-element")
export default class MyTypeScriptElement extends UmbElementMixin(LitElement) {
    #notificationContext?: UmbNotificationContext;

    constructor() {
        super();
        this.consumeContext(UMB_NOTIFICATION_CONTEXT, (_instance) => {
            this.#notificationContext = _instance;
        });
    }

    #onClick = () => {
        this.#notificationContext?.peek("positive", {
            data: { message: "#h5yr" },
        });
    };
```

Update /App_Plugins/my-typescript-extension/src/my-element.ts



```typescript

    render() {
        return html`
            <uui-box headline="Welcome">
                <p>A TypeScript Lit Dashboard</p>
                <uui-button
                    look="primary"
                    label="Click me"
                    @click=${this.#onClick}
                ></uui-button>
            </uui-box>
        `;
    }
}

declare global {
    interface HTMLElementTagNameMap {
        "my-typescript-element": MyTypeScriptElement;
    }
}

```

Create /App_Plugins/my-typescript-extension/umbraco-package.json

```json
{
    "$schema": "../../umbraco-package-schema.json",
    "name": "My TypeScript Extension",
    "version": "0.1.0",
    "extensions": [
        {
            "type": "dashboard",
            "alias": "My.Dashboard.MyTypeScriptExtension",
            "name": "My TypeScript Extension",
            "js": "/App_Plugins/my-typescript-extension/dist/my-typescript-extension.
            "weight": -1,
            "meta": {
                "label": "My TypeScript Extension",
                "pathname": "my-typescript-extension"
            },
            "conditions": [
                {
                    "alias": "Umb.Condition.SectionAlias",
                    "match": "Umb.Section.Content"
                }
            ]
        }
    ]
}
```

Getting started with Kevin Jump!

- Install the template

```
dotnet new install Umbraco.Community.Early.Templates
```

- Create the project

```
dotnet new early.umbracopackage -n TimeDashboard --restore
```

- Run dotnet run in TimeDashboard.Website

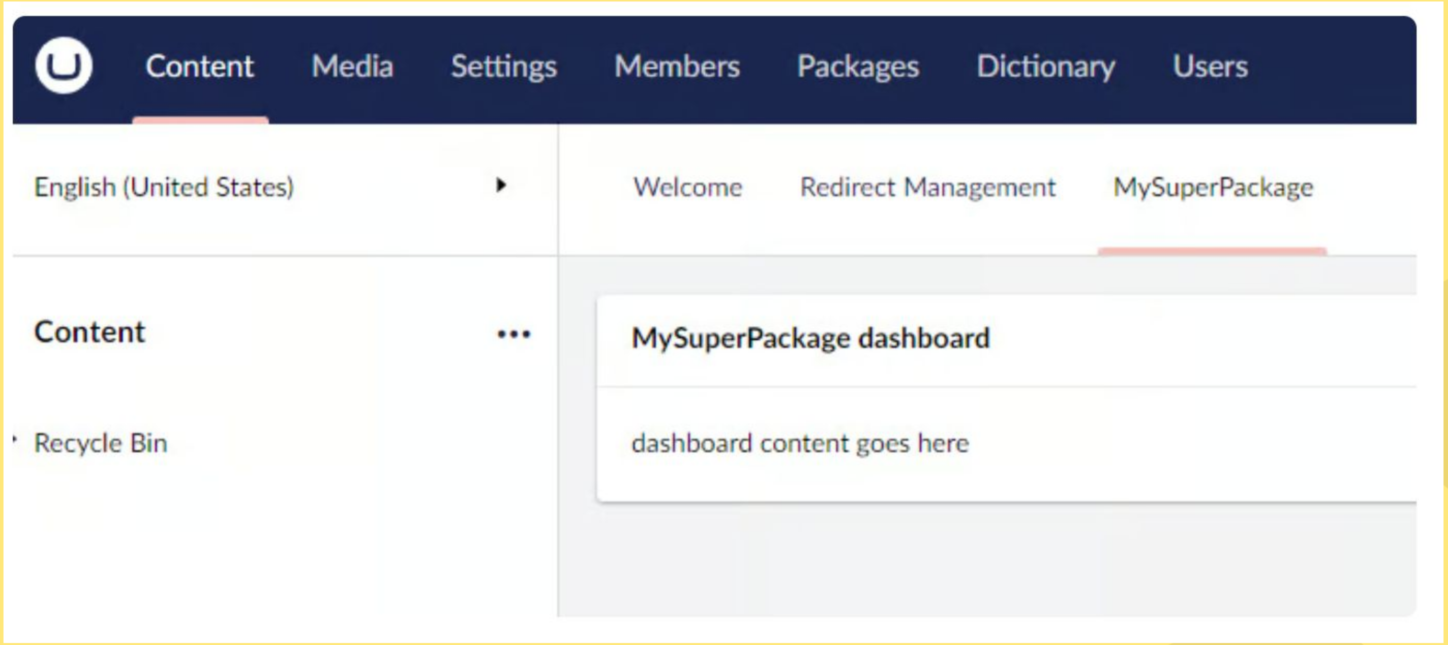
```
cd TimeDashboard.Website

dotnet run
```

https://dev.to/kevinjump/early-adopters-umbraco-package-template-fbh

There’s a very handy repo

https://github.com/KevinJump/TimeDashboard

Getting started with Kevin Jump!



MySuperPackage dashboard

dashboard content goes here

https://dev.to/kevinjump/early-adopters-umbraco-package-template-fbh

Getting started with Kevin Jump!

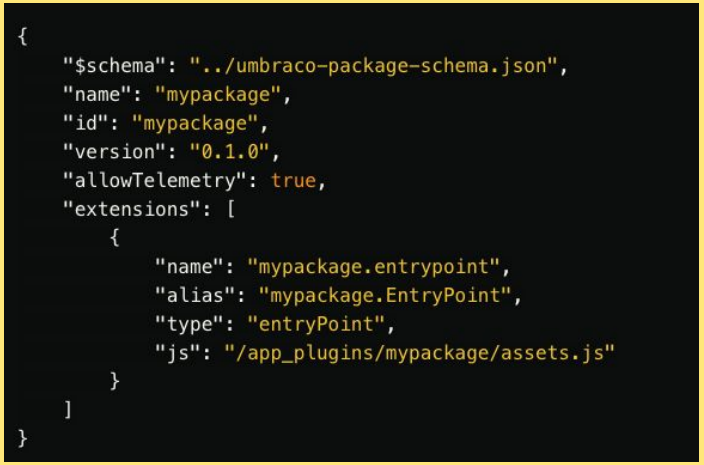
- Build and running the frontend

```
cd TimeDashboard.Client

cd assets

npm run watch
```

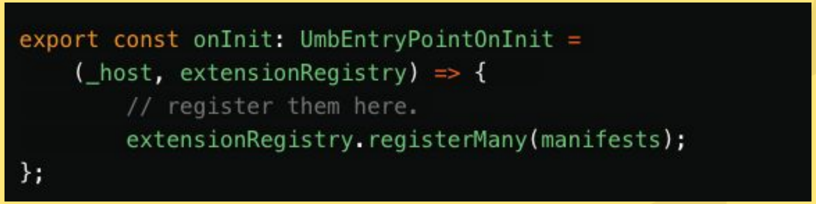
https://dev.to/kevinjump/early-adopters-umbraco-package-template-fbh

Entity points

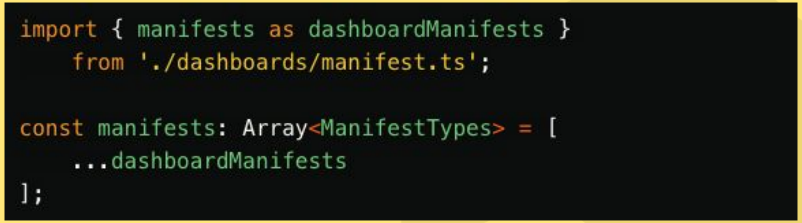
- Default in Umbraco docs is the umbraco-package.json
- But with larger projects you can use a Entry Point extension



```json
{
    "$schema": "../umbraco-package-schema.json",
    "name": "mypackage",
    "id": "mypackage",
    "version": "0.1.0",
    "allowTelemetry": true,
    "extensions": [
        {
            "name": "mypackage.entrypoint",
            "alias": "mypackage.EntryPoint",
            "type": "entryPoint",
            "js": "/app_plugins/mypackage/assets.js"
        }
    ]
}
```



```ts
export const onInit: UmbEntryPointOnInit =
    (_host, extensionRegistry) => {
        // register them here.
        extensionRegistry.registerMany(manifests);
};
```



```ts
import { manifests as dashboardManifests }
    from './dashboards/manifest.ts';

const manifests: Array<ManifestTypes> = [
    ...dashboardManifests
];
```

https://dev.to/kevinjump/early-adopters-guide-entry-points-50jj

Communicating with the server!

C#

- Write some c# controllers
- Get OpenAPI / Swagger working
- Generate some typescript models

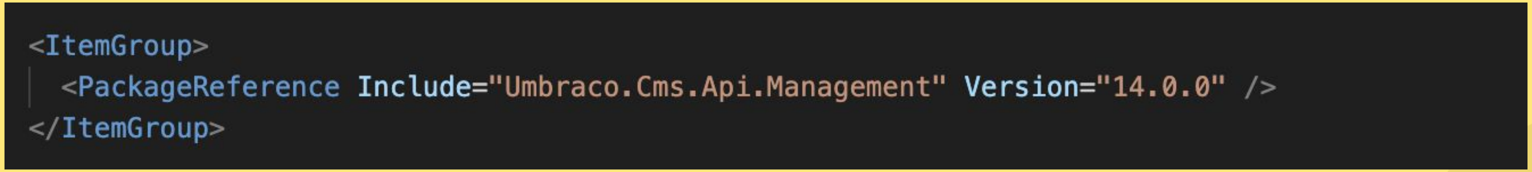
Typescript

- Add data source
- Add repository
- Create context
- Add authentication
- Wire up in a manifest file

https://dev.to/kevinjump/early-adopters-guide-to-umbraco-v14-packages-communicating-with-the-server-part-1-38lb

Finishing project setup

Add a package to the TimeDashboard.Client/TimeDashboard.Client.csproj file

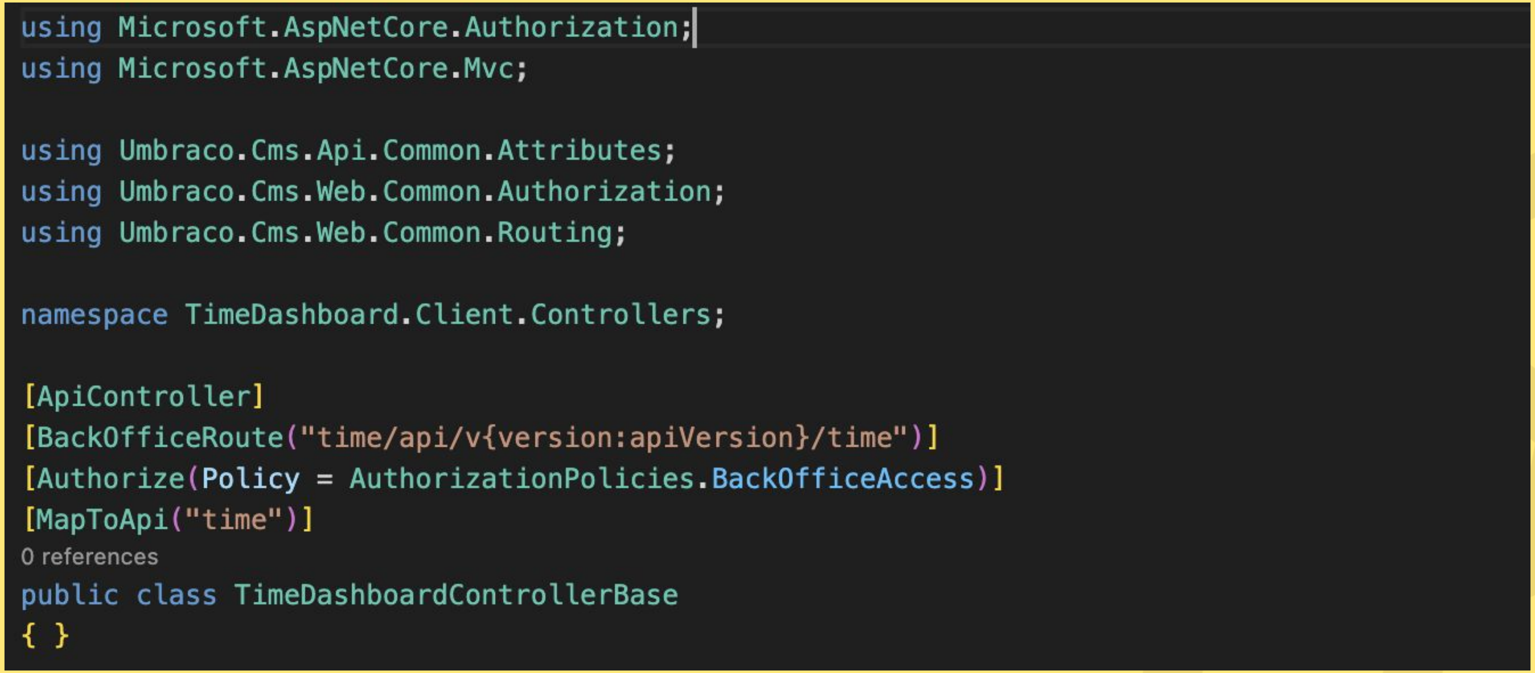


```xml
<ItemGroup>
  <PackageReference Include="Umbraco.Cms.Api.Management" Version="14.0.0" />
</ItemGroup>
```

https://dev.to/kevinjump/early-adopters-guide-to-umbraco-v14-packages-communicating-with-the-server-part-1-38lb

Base controller

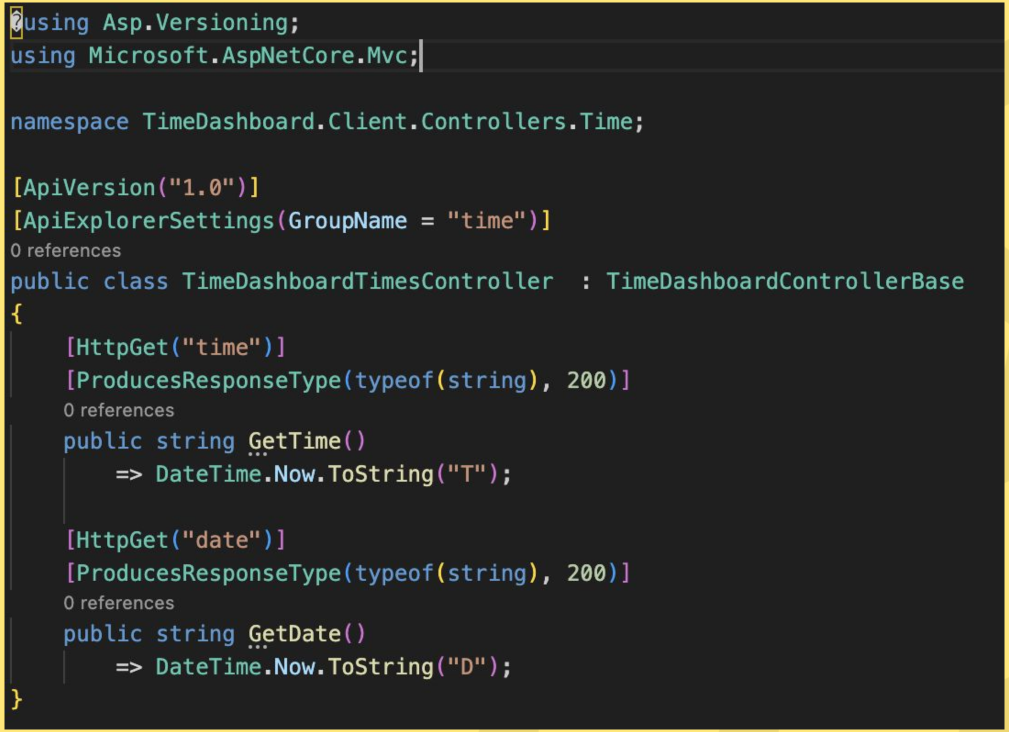
Create a base controller - TimeDashboard.Client/Controllers/TimeDashboardBaseController.cs



```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

using Umbraco.Cms.Api.Common.Attributes;
using Umbraco.Cms.Web.Common.Authorization;
using Umbraco.Cms.Web.Common.Routing;

namespace TimeDashboard.Client.Controllers;

[ApiController]
[BackOfficeRoute("time/api/v{version:apiVersion}/time")]
[Authorize(Policy = AuthorizationPolicies.BackOfficeAccess)]
[MapToApi("time")]
0 references
public class TimeDashboardControllerBase
{ }
```

Controller

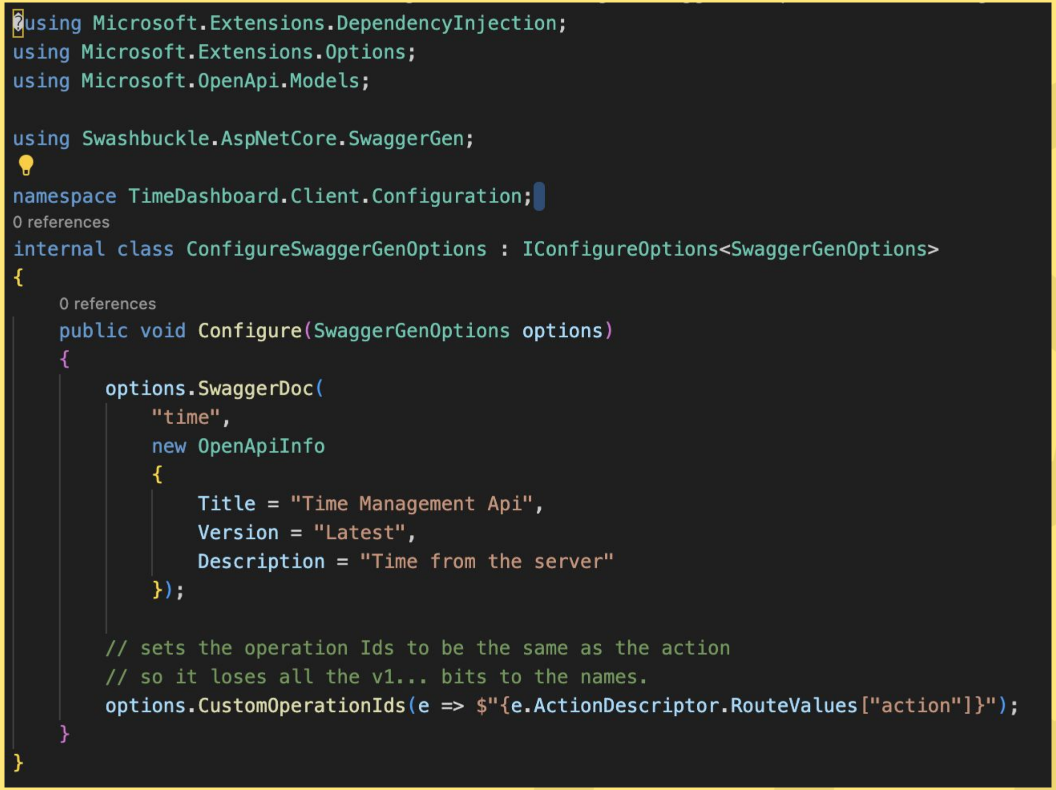
Create a times controller - TimeDashboard.Client/Controllers/Time/TimeDashboardTimeController.cs



```
using Asp.Versioning;
using Microsoft.AspNetCore.Mvc;

namespace TimeDashboard.Client.Controllers.Time;

[ApiVersion("1.0")]
[ApiExplorerSettings(GroupName = "time")]
0 references
public class TimeDashboardTimesController  : TimeDashboardControllerBase
{
    [HttpGet("time")]
    [ProducesResponseType(typeof(string), 200)]
    0 references
    public string GetTime()
        => DateTime.Now.ToString("T");

    [HttpGet("date")]
    [ProducesResponseType(typeof(string), 200)]
    0 references
    public string GetDate()
        => DateTime.Now.ToString("D");
}
```

Open API config

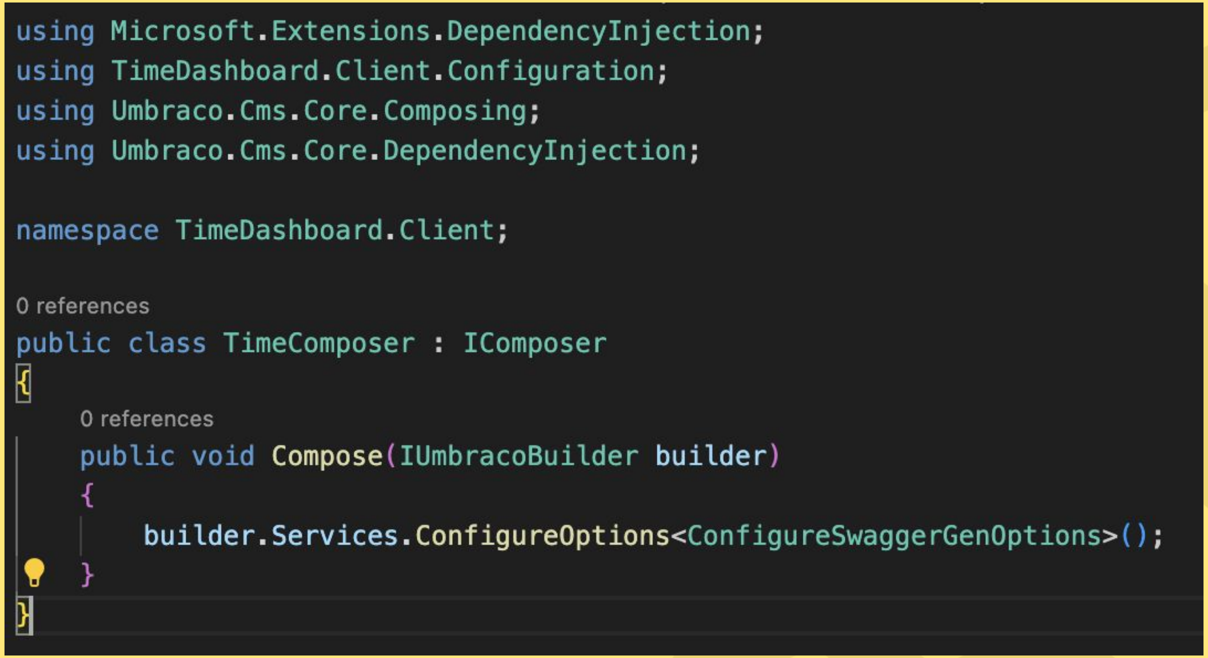
Add Swagger configuration - TimeDashboard.Client/Configuration /ConfigureSwaggerGenOptions.cs



```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Options;
using Microsoft.OpenApi.Models;

using Swashbuckle.AspNetCore.SwaggerGen;

namespace TimeDashboard.Client.Configuration;
0 references
internal class ConfigureSwaggerGenOptions : IConfigureOptions<SwaggerGenOptions>
{
    0 references
    public void Configure(SwaggerGenOptions options)
    {
        options.SwaggerDoc(
            "time",
            new OpenApiInfo
            {
                Title = "Time Management Api",
                Version = "Latest",
                Description = "Time from the server"
            });

        // sets the operation Ids to be the same as the action
        // so it loses all the v1... bits to the names.
        options.CustomOperationIds(e => $"{e.ActionDescriptor.RouteValues["action"]}");
    }
}
```

Composing

Add composer - TimeDashboard.Client/ TimeComposer.cs



```
using Microsoft.Extensions.DependencyInjection;
using TimeDashboard.Client.Configuration;
using Umbraco.Cms.Core.Composing;
using Umbraco.Cms.Core.DependencyInjection;

namespace TimeDashboard.Client;

0 references
public class TimeComposer : IComposer
{
    0 references
    public void Compose(IUmbracoBuilder builder)
    {
        builder.Services.ConfigureOptions<ConfigureSwaggerGenOptions>();
    }
}
```

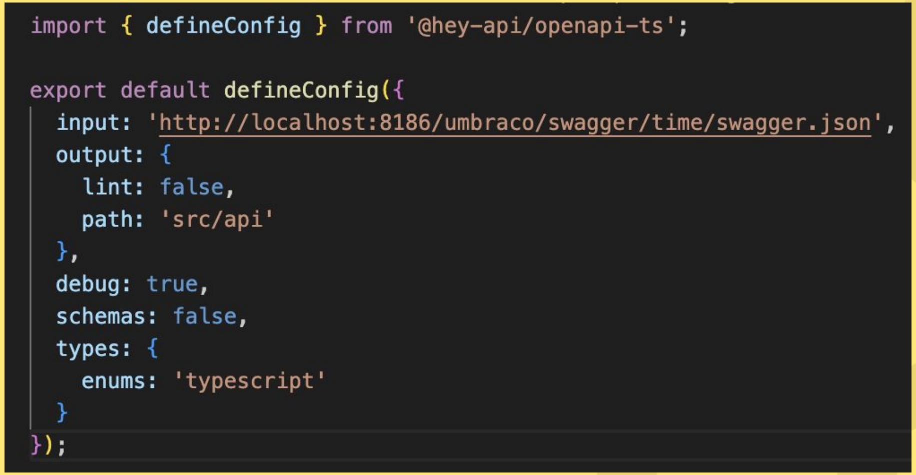
Model generation

- Generate some typescript models

  npm install -–save-dev @hey-api/openapi-ts

  Add to TimeDashboard.Client/assets/package.json scripts - “generate”: “openapi-ts”

- Create TimeDashboard.Client /assets/openapi-ts.config.js



```js
import { defineConfig } from '@hey-api/openapi-ts';

export default defineConfig({
  input: 'http://localhost:8186/umbraco/swagger/time/swagger.json',
  output: {
    lint: false,
    path: 'src/api'
  },
  debug: true,
  schemas: false,
  types: {
    enums: 'typescript'
  }
});
```

The frontend call stack

- **Resources** are the things that actually go fetch the data, in the case of the back office, these are the bits doing the http requests.

- **Stores** are where the data is stored

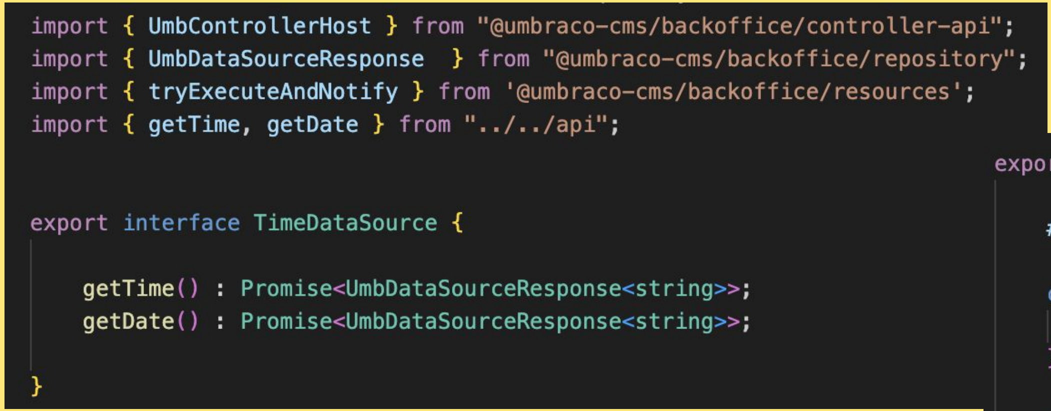
- **Repositories** handle the first part of getting things, they abstract away how the data is stored in your app

- **Context** is like a service, it's go the method you want to get things in your dashboard, etc. it also stores some values, which you can 'observe' for changes than might happen elsewhere.

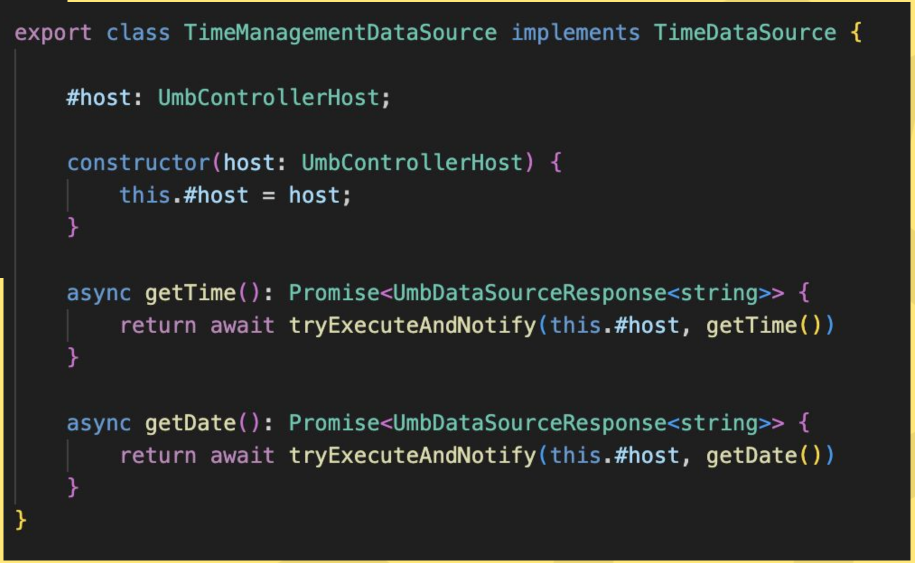
https://dev.to/kevinjump/early-adopters-guide-to-umbraco-v14-packages-communicating-with-the-server-part-1-38lb

Add a time store (datasource)

- Create TimeDashboard.Client/assets/src/repository/sources/time.datasource.ts



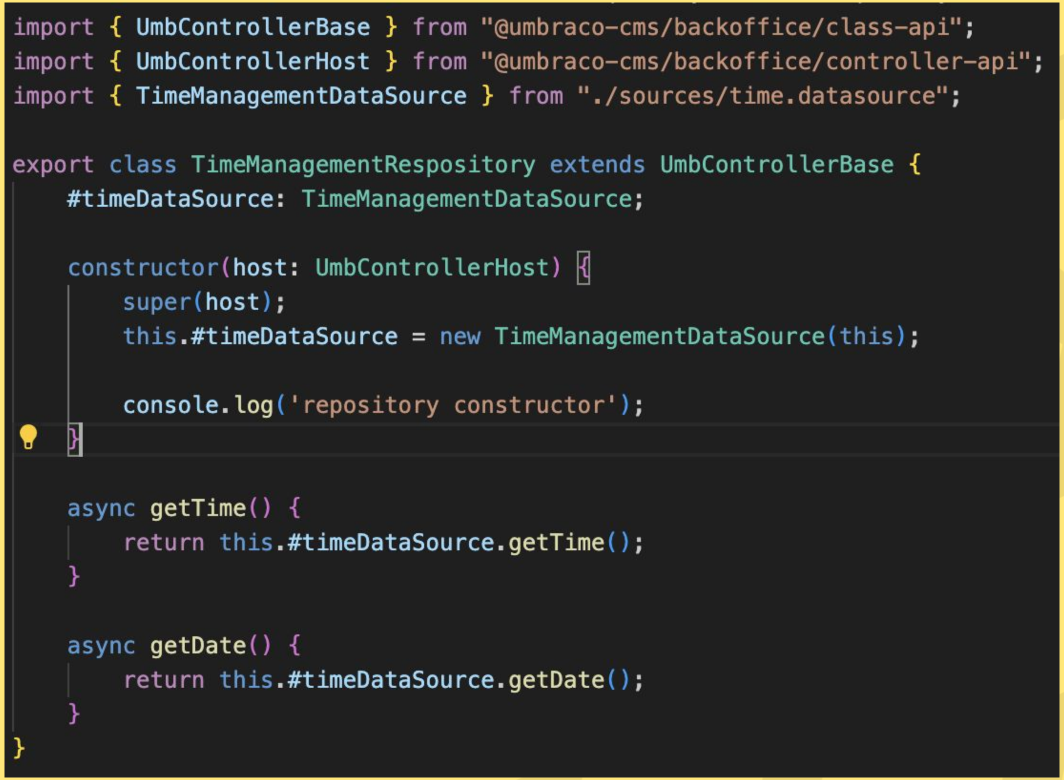
```
import { UmbControllerHost } from "@umbraco-cms/backoffice/controller-api";
import { UmbDataSourceResponse  } from "@umbraco-cms/backoffice/repository";
import { tryExecuteAndNotify } from '@umbraco-cms/backoffice/resources';
import { getTime, getDate } from "../../api";


export interface TimeDataSource {

    getTime() : Promise<UmbDataSourceResponse<string>>;
    getDate() : Promise<UmbDataSourceResponse<string>>;

}
```



```
export class TimeManagementDataSource implements TimeDataSource {

    #host: UmbControllerHost;

    constructor(host: UmbControllerHost) {
        this.#host = host;
    }

    async getTime(): Promise<UmbDataSourceResponse<string>> {
        return await tryExecuteAndNotify(this.#host, getTime())
    }

    async getDate(): Promise<UmbDataSourceResponse<string>> {
        return await tryExecuteAndNotify(this.#host, getDate())
    }
}
```

Add a time repository

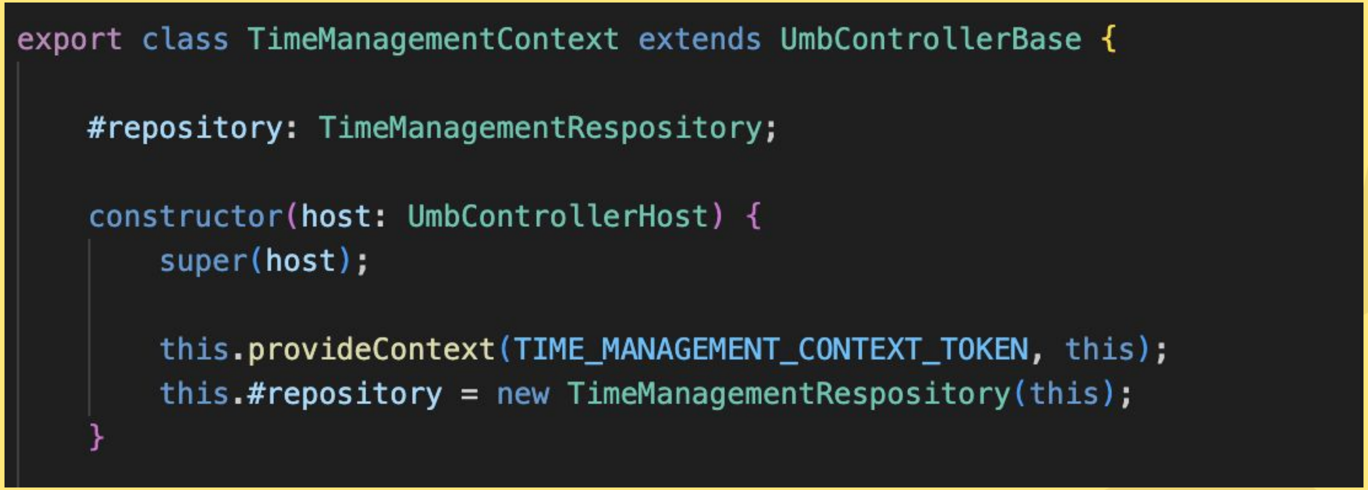
- Create TimeDashboard.Client/assets/src/repository/time.repository.ts



```typescript
import { UmbControllerBase } from "@umbraco-cms/backoffice/class-api";
import { UmbControllerHost } from "@umbraco-cms/backoffice/controller-api";
import { TimeManagementDataSource } from "./sources/time.datasource";

export class TimeManagementRespository extends UmbControllerBase {
    #timeDataSource: TimeManagementDataSource;

    constructor(host: UmbControllerHost) {
        super(host);
        this.#timeDataSource = new TimeManagementDataSource(this);

        console.log('repository constructor');
    }

    async getTime() {
        return this.#timeDataSource.getTime();
    }

    async getDate() {
        return this.#timeDataSource.getDate();
    }
}
```

Create time management context

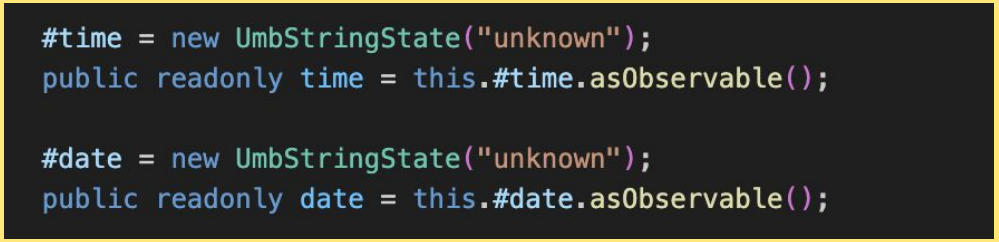
- Create TimeDashboard.Client/assets/src/context/time.context.ts



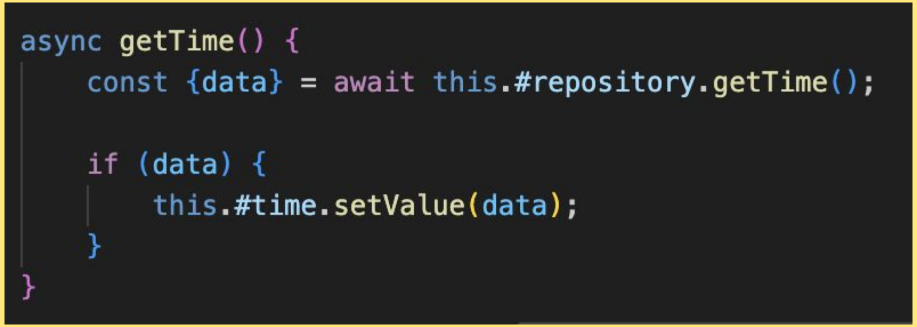
```
export class TimeManagementContext extends UmbControllerBase {

    #repository: TimeManagementRespository;

    constructor(host: UmbControllerHost) {
        super(host);

        this.provideContext(TIME_MANAGEMENT_CONTEXT_TOKEN, this);
        this.#repository = new TimeManagementRespository(this);
    }
```

It’s best to copy and paste this this directly from the repo

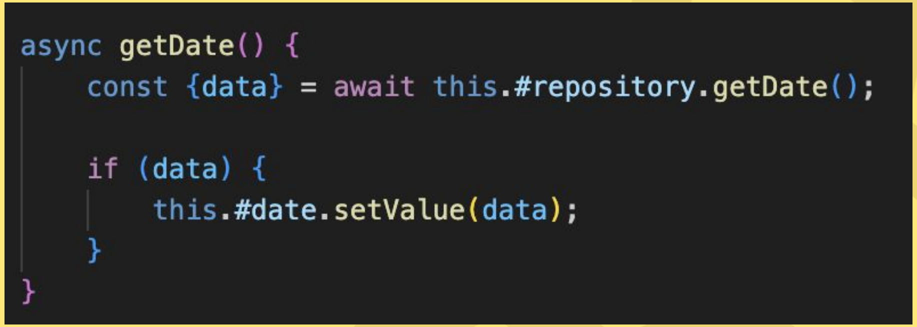
Context method calls



```
#time = new UmbStringState("unknown");
public readonly time = this.#time.asObservable();

#date = new UmbStringState("unknown");
public readonly date = this.#date.asObservable();
```

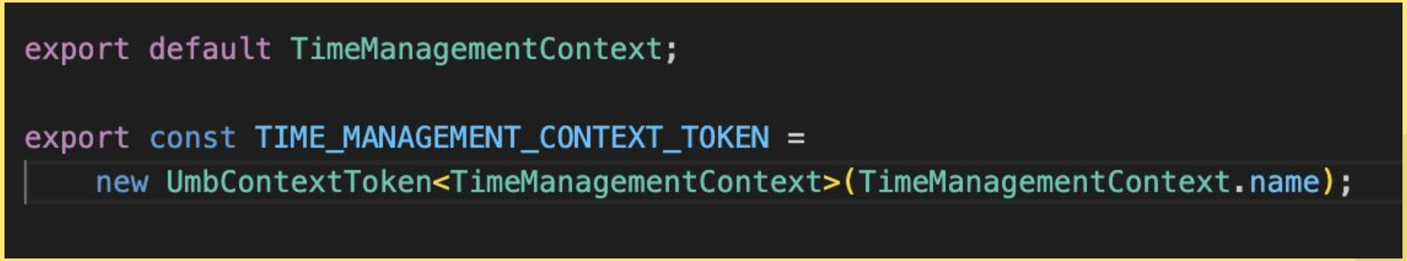


```
async getTime() {
    const {data} = await this.#repository.getTime();

    if (data) {
        this.#time.setValue(data);
    }
}
```



```
async getDate() {
    const {data} = await this.#repository.getDate();

    if (data) {
        this.#date.setValue(data);
    }
}
```

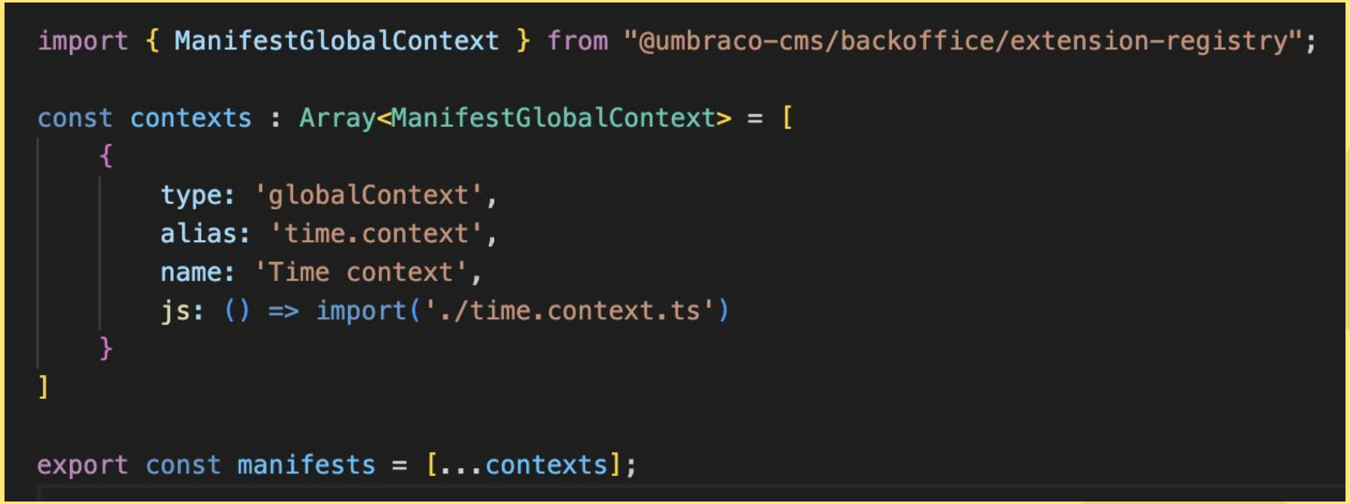
Context export instance and token



```
export default TimeManagementContext;

export const TIME_MANAGEMENT_CONTEXT_TOKEN =
    new UmbContextToken<TimeManagementContext>(TimeManagementContext.name);
```

Add local context manifest

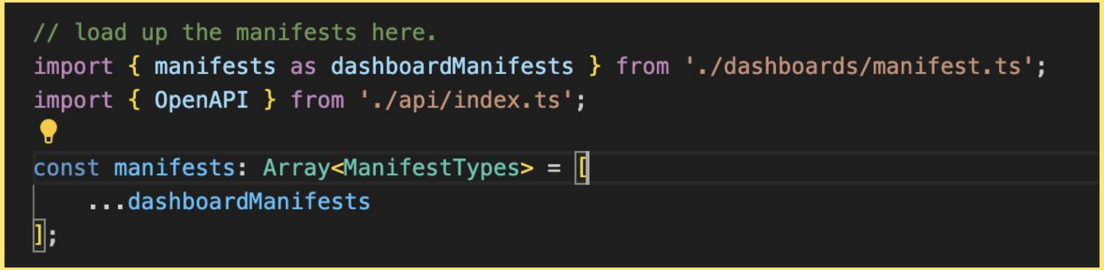
- Create TimeDashboard.Client/assets/src/context/manifest.ts



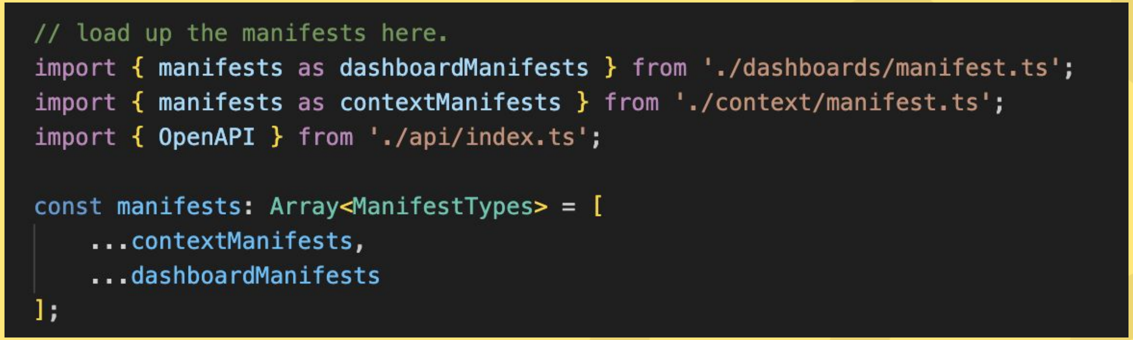
```
import { ManifestGlobalContext } from "@umbraco-cms/backoffice/extension-registry";

const contexts : Array<ManifestGlobalContext> = [
    {
        type: 'globalContext',
        alias: 'time.context',
        name: 'Time context',
        js: () => import('./time.context.ts')
    }
]

export const manifests = [...contexts];
```

Add to global manifest

- Update TimeDashboard.Client/assets/src/index.ts from this to this



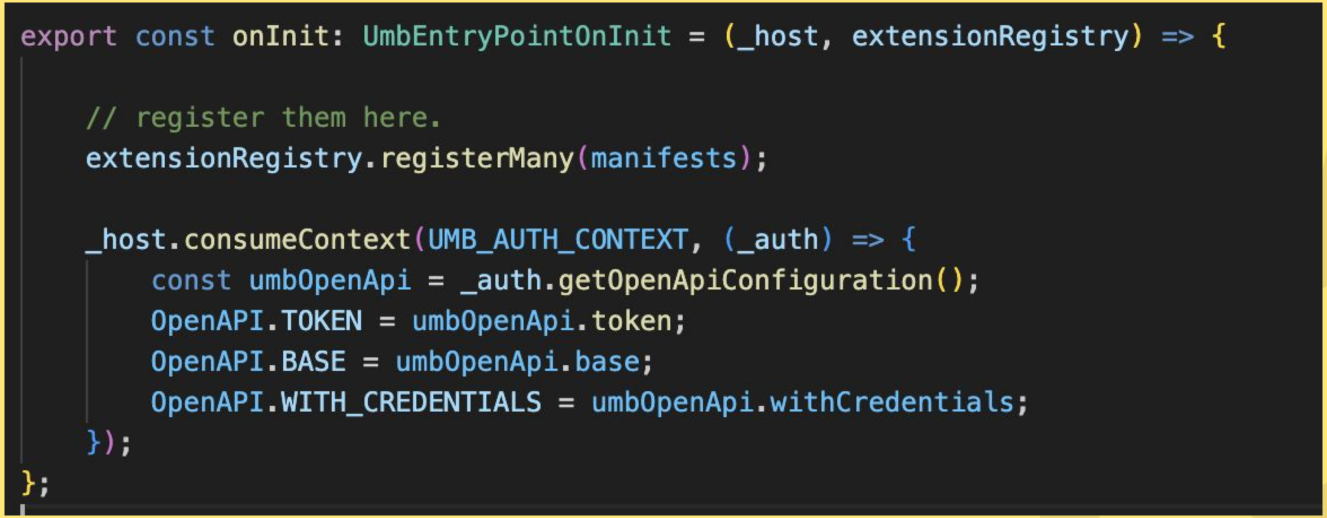
```
// load up the manifests here.
import { manifests as dashboardManifests } from './dashboards/manifest.ts';
import { OpenAPI } from './api/index.ts';

const manifests: Array<ManifestTypes> = [
    ...dashboardManifests
];
```



```
// load up the manifests here.
import { manifests as dashboardManifests } from './dashboards/manifest.ts';
import { manifests as contextManifests } from './context/manifest.ts';
import { OpenAPI } from './api/index.ts';

const manifests: Array<ManifestTypes> = [
    ...contextManifests,
    ...dashboardManifests
];
```

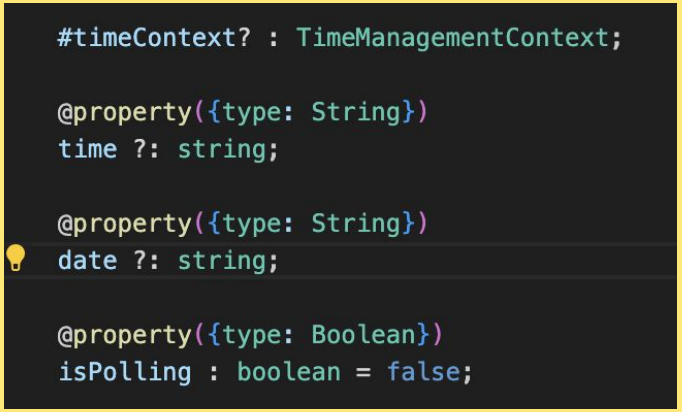
Add authentication to global manifest

- Add the _host.consumeContext to onInit in TimeDashboard.Client/assets/src/index.ts

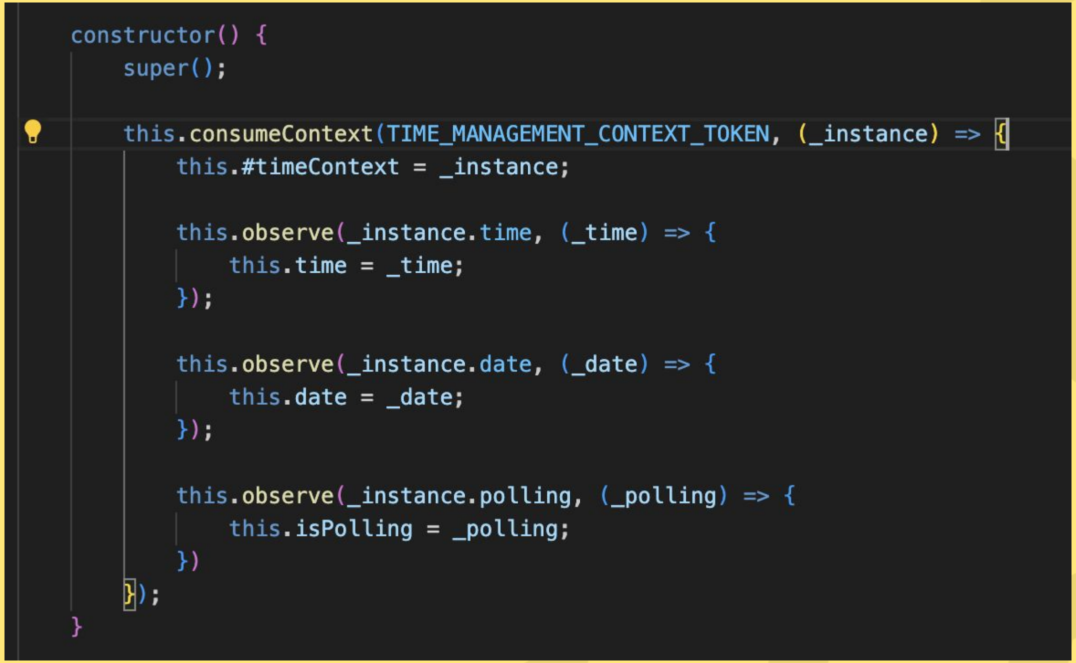


```
export const onInit: UmbEntryPointOnInit = (_host, extensionRegistry) => {

    // register them here.
    extensionRegistry.registerMany(manifests);

    _host.consumeContext(UMB_AUTH_CONTEXT, (_auth) => {
        const umbOpenApi = _auth.getOpenApiConfiguration();
        OpenAPI.TOKEN = umbOpenApi.token;
        OpenAPI.BASE = umbOpenApi.base;
        OpenAPI.WITH_CREDENTIALS = umbOpenApi.withCredentials;
    });
};
```

Update the dashboard element - add context

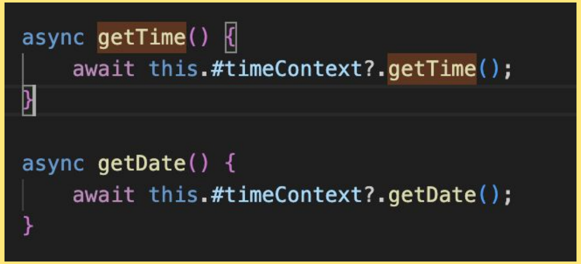


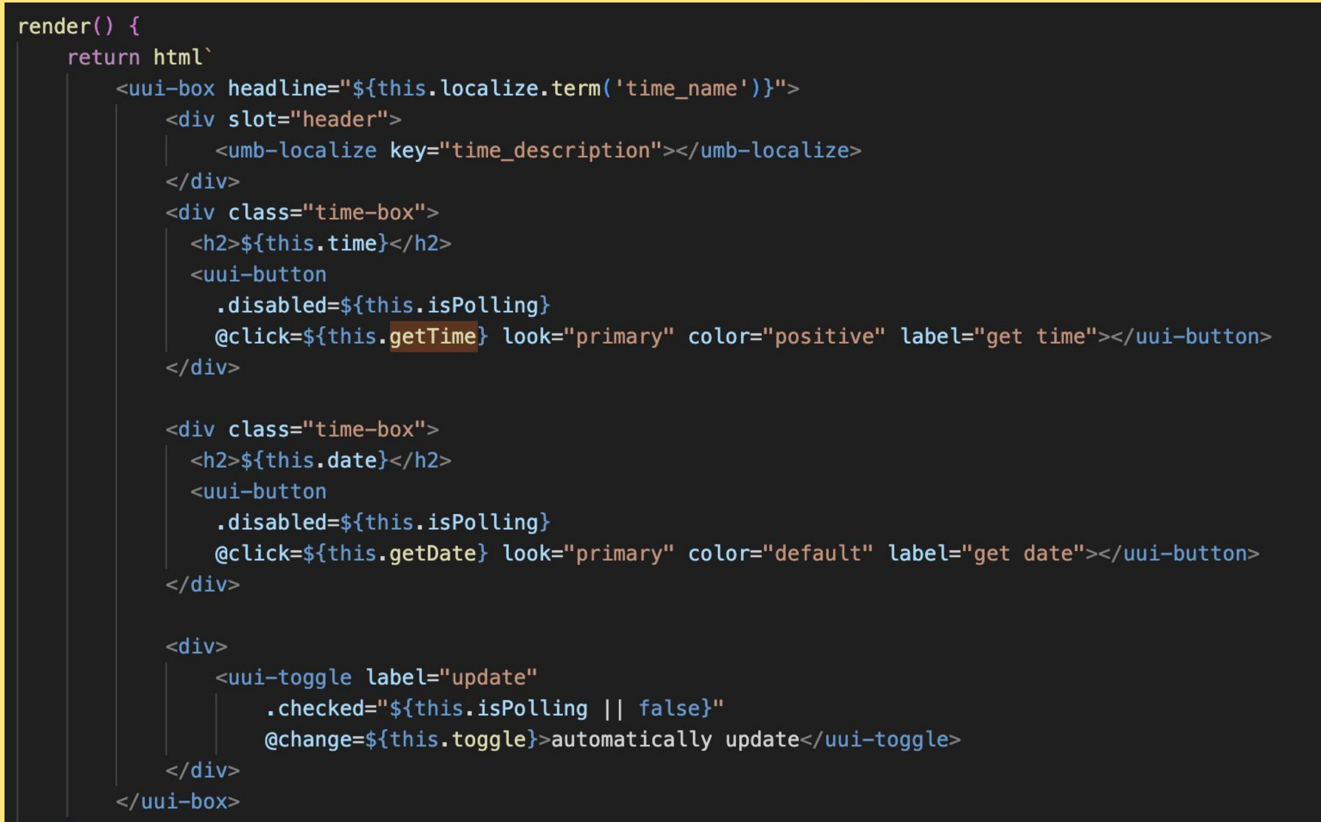
```
#timeContext? : TimeManagementContext;

@property({type: String})
time ?: string;

@property({type: String})
date ?: string;

@property({type: Boolean})
isPolling : boolean = false;
```

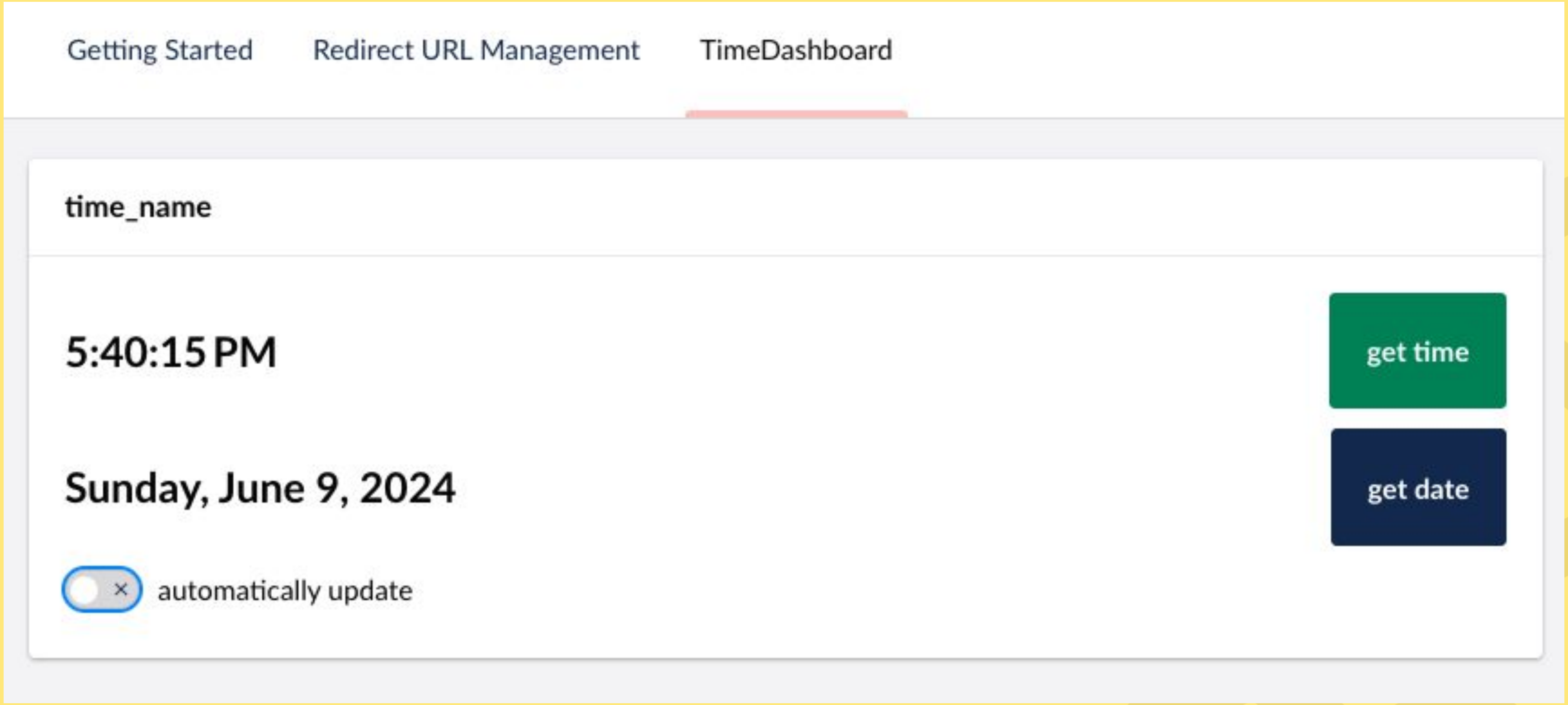


```
constructor() {
    super();

    this.consumeContext(TIME_MANAGEMENT_CONTEXT_TOKEN, (_instance) => {
        this.#timeContext = _instance;

        this.observe(_instance.time, (_time) => {
            this.time = _time;
        });

        this.observe(_instance.date, (_date) => {
            this.date = _date;
        });

        this.observe(_instance.polling, (_polling) => {
            this.isPolling = _polling;
        })
    });
}
```

Update the dashboard element - add markup & event handlers


```
async getTime() {
    await this.#timeContext?.getTime();
}

async getDate() {
    await this.#timeContext?.getDate();
}
```


```
render() {
    return html`
        <uui-box headline="${this.localize.term('time_name')}">
            <div slot="header">
                <umb-localize key="time_description"></umb-localize>
            </div>
            <div class="time-box">
              <h2>${this.time}</h2>
              <uui-button
                .disabled=${this.isPolling}
                @click=${this.getTime} look="primary" color="positive" label="get time"></uui-button>
            </div>

            <div class="time-box">
              <h2>${this.date}</h2>
              <uui-button
                .disabled=${this.isPolling}
                @click=${this.getDate} look="primary" color="default" label="get date"></uui-button>
            </div>

            <div>
                <uui-toggle label="update"
                    .checked="${this.isPolling || false}"
                    @change=${this.toggle}>automatically update</uui-toggle>
            </div>
        </uui-box>
```



time_name

5:40:15 PM

Sunday, June 9, 2024

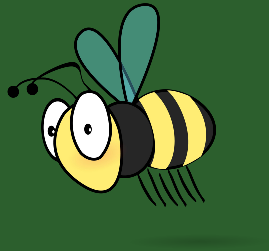
automatically update

Branches for reference

https://github.com/whitter/umbraco-14-workshop

umbraco
manchester

thank you



Phil Whittaker
HiFi

Rachel Breeze
Nexer

Jon Whitter
Cantarus

organisers