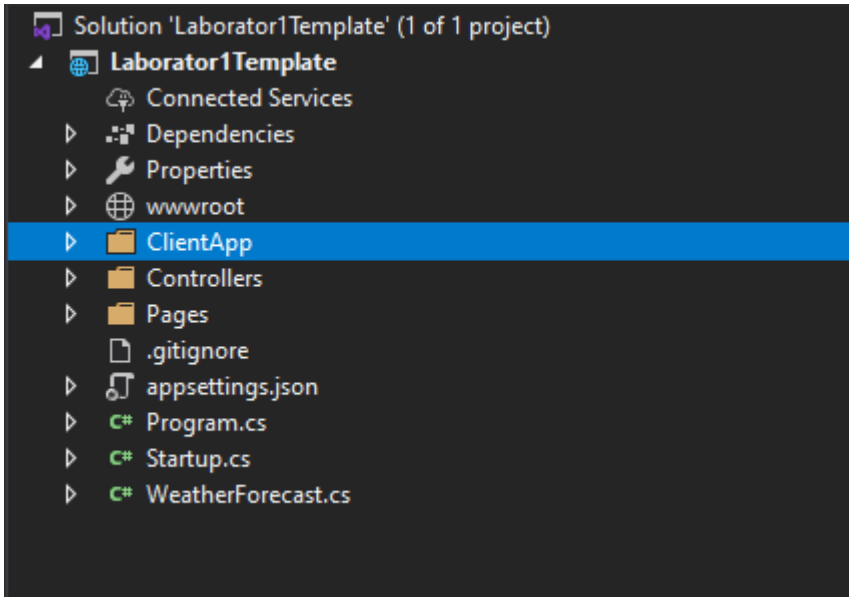
Laborator 1

Intro in .NET Core, Web APIs & so on

Hai sa ne jucam putin cu .NET. Deschidem Visual Studio si creem un proiect nou. Noi o sa lucram cu aplicatii **ASP .NET Core Web Application** in **C#**, deci acesta este template-ul de care vom avea nevoie, dam ce nume sugestiv ne vine pe loc, continuam cu ce template am vrea pentru frontend (sau daca nu vrem frontend), deci vom alege **Angular** (noi vom lucra, in viitor, cu Angular; daca intentionati sa folositi React, go for it, doar ca nu va fi in suportul cursului). Am creat proiectul, ne putem juca 😄

Cam asa ar trebui sa arate solutia noastra:



Noi vom lucra pe baza acestui template, vom pleca de la el, il vom dezvolta, vom adauga chestii smechere si il vom face sa lucreze ca o aplicatie calumea, completa.

Cate ceva despre .NET Core, utile de-a lungul laboratorului

Functioneaza ca o aplicatie de consola; motiv pentru care are o functie Main (public static void Main(args)) care este apelata la pornire, gasita in fisierul

Program.cs. Ea nu face decat sa ridice un Host si sa specifice clasa **Startup** pentru configurari; isi face treaba out-of-the-box, nu avem treaba cu ea.

Clasa **Startup** este mai interesanta putin; aici se gasesc toate configurarile aplicatiei. Este inima solutiei; de exemplu, cand vom avea nevoie sa adaugam o librarie noua, aici vom intra; cum vom vrea sa evoluam aplicatia si sa-i mai aduagam o chestie, doua, aici vom mai avea treaba.

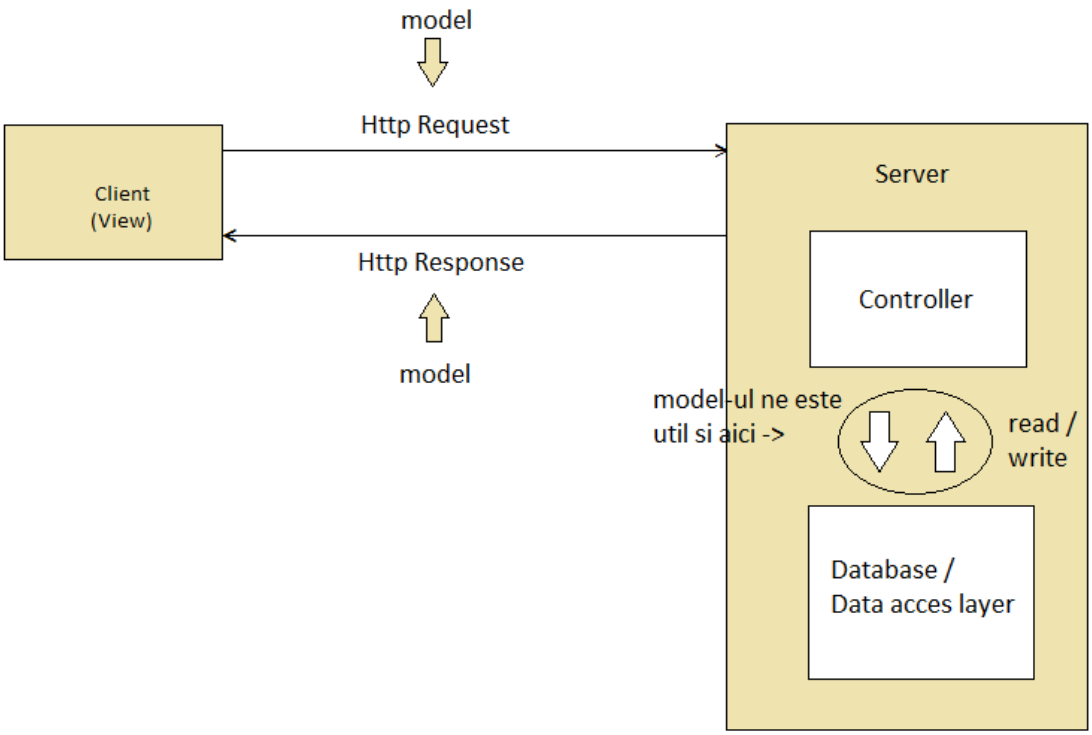
Core, fata de invechitul Framework, este mai lightweight, si portabil pe Linux / Mac. Asta datorita faptului ca toate dependintele (inclusiv cele din librariile de Windows) au fost puse intr-un dll, toate la un loc, si apoi acel dll spart in mai multe dll-uri mai mici. Astfel, aplicatia vin din cutie cu minimul necesar pentru a rula, iar fiecare nevoie individuala se gaseste in alt dll, il alta librarie, pe care ti-o poti trage cand e necesar. Astfel, apar **NuGet**-urile, si **NuGet Package Manager**

NuGet = librarie (basically); continue mai multe DLL-uri, care au si ele niste dependinte in spate, etc. Iar NuGet Package Manager este sistemul prin care se gestioneaza librarii in ecosistemul .NET; sort of npm (prin NuGet Package Manager vom lua librarii pe care le vom adauga in Startup-ul de care vorbeam mai sus). Pentru a accesa, avem in bara de sus Tools -> NuGet Package Manager.

MVC si aplicatiile noastre de toate zilele

O sa presupun ca ati mai auzi / ati mai lucrat cu MVC. Design pattern-ul de **M**odel **V**iew **C**ontroller, popular mai peste tot intr-o forma sau alta (MVVM in aplicatii complexe, MVP – android, etc). Impartirea pe date (model), cum sunt ele afisate (view) si logica lor de manipulare (controller) este destul de intuitiva si aplicabila peste tot.

Noi vom face o mica adaptare: cum vom folosi un client-server arhitecture, controller-ul va fi pe server, modelul pe amandoua (fiind „contractul" intre cele doua), iar view-ul pe client. Dar mai usor, tineti minte doar schema asta:



Schema de mai sus o sa ne fie de folos mai tot laboratorul 😄 Pare mai complicata decat este, va asigur.

Astfel, ajungem in Web API Controllers.

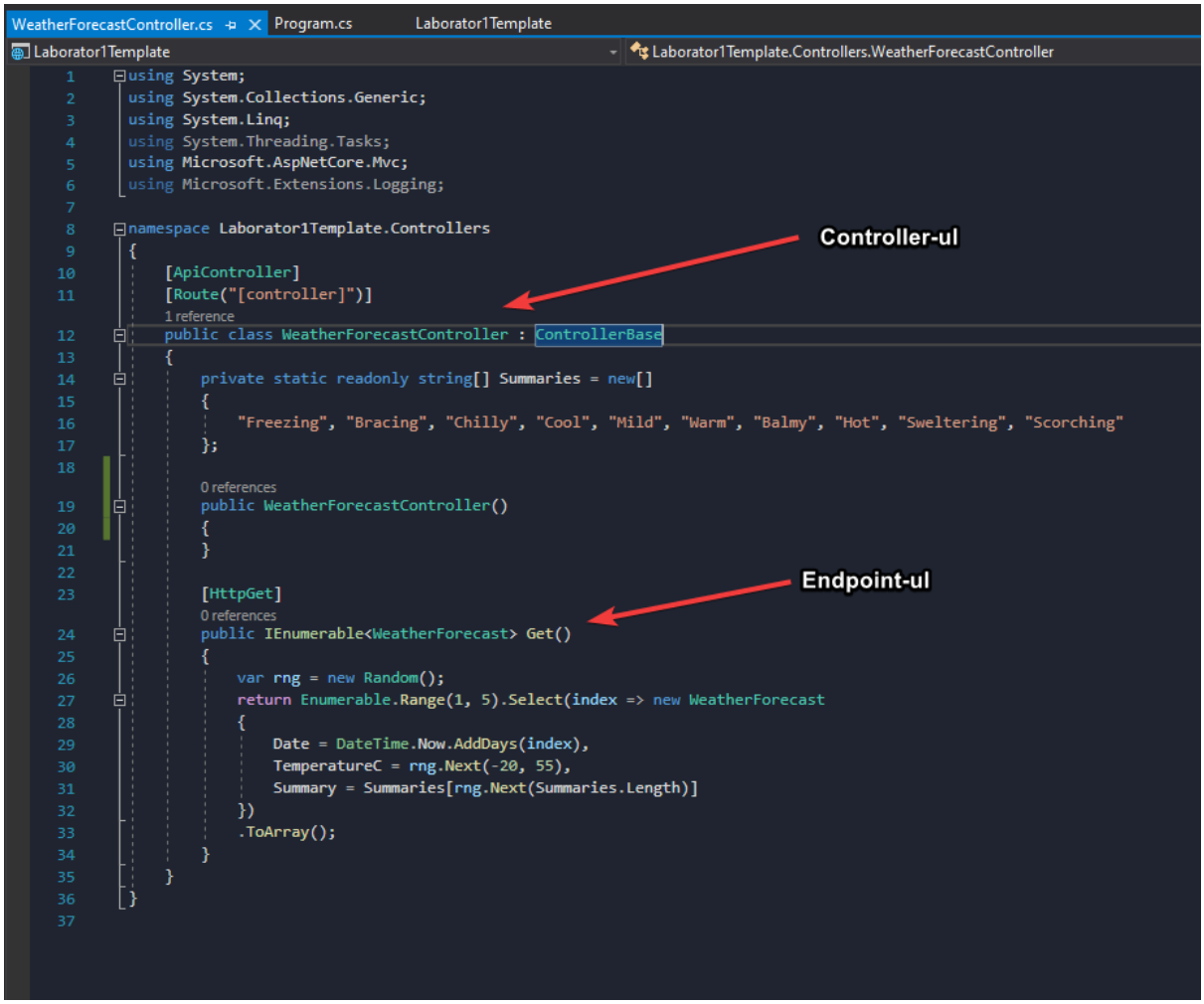
Poveste / Teorie (😵): In .NET MVC clasic, Controller-ul returna direct View-ul, randat pe server in functie de model-ul respectiv (returna un cod html care se punea pe pagina de client). Cum, in a noastra utilizare, avem codul de client separat de cel de server, aceasta abordare nu mai functioneaza, motiv pentru care Controller-ul va returna direct Modelul, datele, sub format JSON; el va avea rolul unui Web API.

Ce trebuie sa stiti despre ideea de API: este o interfata (din punctul de vedere a clientului) ce ofera un set de operatii catre alte programatori.

Controllere noastre, vor avea un rol de Web API, pentru ca ele vor oferi anumite operatii, de care partea de client se va folosi. Termenul pentru aceste operatii in cazul unui API Controller este de **API endpoint**. Endpoint, cu sensul de punct de

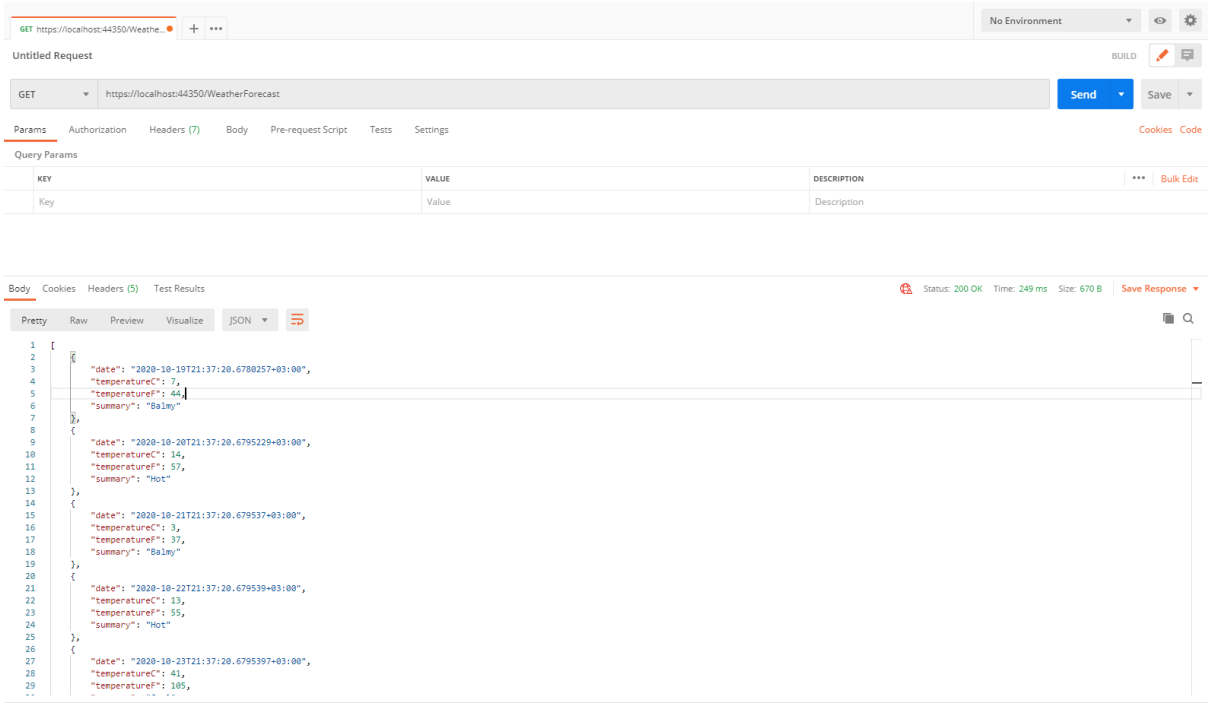
capat al unei comunicari (pentru client, este capatul de final unde poate merge pentru date, pentru server, este capatul de inceput pentru a oferi date).

Un exemplu ar fi chiar Controller-ul implicit pe care ni-l da Template-ul:



Acesta poate fi accesat din afara, si este calea de comunicare cu server-ul. Asa se pot deschide cai de comunicare (date de endpoint-uri) din partea aplicatiei.

Pentru a test API-uri, recomand Postman. Cand ulam proiectul, puteam lua url-ul din browser, il punem in postman, completam cu ruta catre endpoint-ul nostru (/controller/actionName de obicei – actionName este dat in paranteze langa verb-ul endpoint-ului, de ex HttpGet(„weatherForecast”)). Un exemplu de request si response-ul venit din aplicatia noastra ar fi:



Acesta poate fi accesat din afara, si este calea de comunicare cu server-ul. Asa se pot deschide cai de comunicare (date de endpoint-uri) din partea aplicatie.

Ce ar mai fi de specificat:

- CRUD: Create, read, update, delete. Similar, exista Http Verbs; cele mai populare sunt POST, GET, PUT, PATCH, DELETE, care s-ar traduce ca Create, Read, Update, Partial update, Delete(surprise); fiecarui endpoint ii este atasat un Verb, iar requestul din client va trebui sa corespunda.
- Endpoint-urile vor returna un status code; pe langa obiectul / mesajul de care este nevoie, un endpoint va returna si un status code, ce va indica ce s-a intamplat cu respectivul request; cele mai intalnite sunt: **200** (Ok), **404**(Not Found), **400**(Bad Request), **401**(Unauthorized), **405**(Method not allowd – asta este legat de HttpVerb pus gresit) si **500** (Internal Server Error); pentru simplitate, in .NET cand dam return, ne putem folosi de niste metode venite din ControllerBase, intuitive (ex: return Ok(model), return NotFound(model) etc)

O sugestie: cand nu o sa mearga ceva bine, si sigur se va intampla sa fie un null unde nu va asteptati, use debugging. Duce-ti-va pe linia care cauzeaza probleme, apasati F9 (sau click la inceput de rand) pentru a pune un breakpoint, aplicatia se opreste acolo si dati hover pe ce variabile sunt, vedeti ce este null; de acolo va mai puteti juca cu debugger-ul pana gasiti solutia (F5- run pana la final sau urmatorul breakpoint; F10 – run doar o operatie, dupa asteapta input again; F11 – debugger-ul intra in metoda apelata la acest pas).

De aici, va puteti juca; cream un Controller nou (dr mouse pe Controllers -> Add Controller), noi vom folosi Api Controller Empty, dar va incurajez sa va jucati cu fiecare in parte, give it a name, and start coding 😉. Hai sa incercam, fiecare, sa facem un Controller cu operatiunile CRUD asupra unui obiect ales de noi. Cum nu avem inca baza de date, mock-uim, si rezolvam urmatorul laborator si asta. Incercati, schimbati, creati, rulati, daca nu merge, sunt probleme sau sunt intrebari, ask me.