

Adding an API Using ASP.NET Core



Gill Cleeren

CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be



Overview



Creating the API project

Transitioning from view services to MediatR

Deciding which objects to return

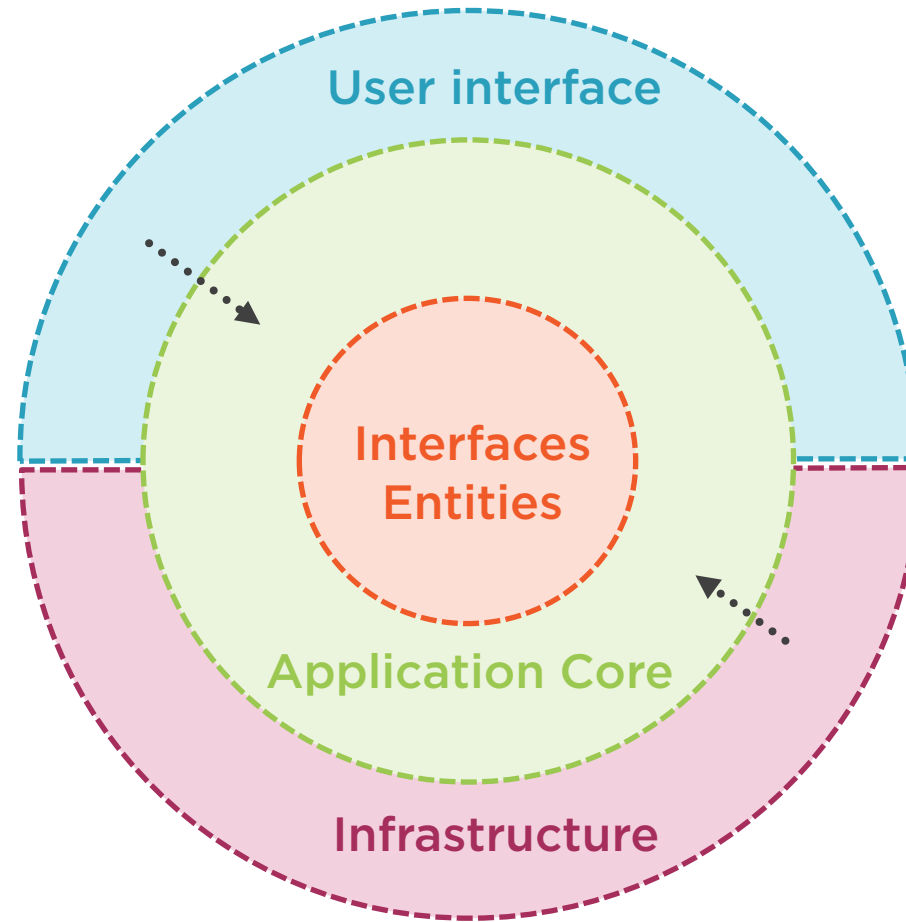
Exposing the API functionality using Swagger



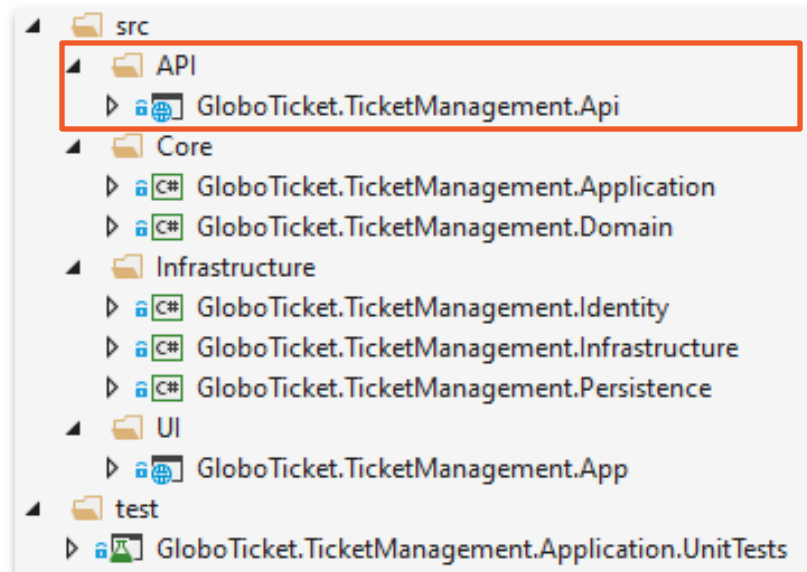
Creating the API Project

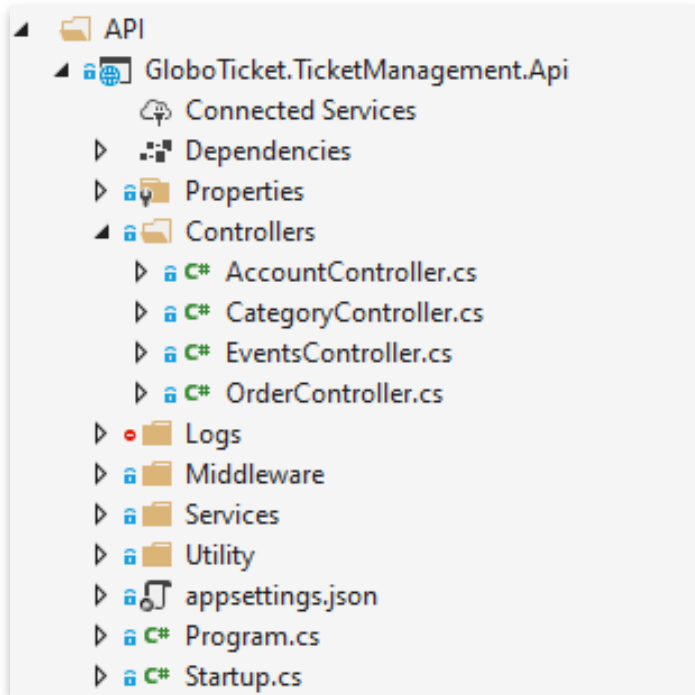


Bringing in the User Interface



The API in Our Architecture





ASP.NET Core API

Startup configuration

- Calls to other project ServiceCollection extensions

References to other projects

- Mostly because of DI



Demo



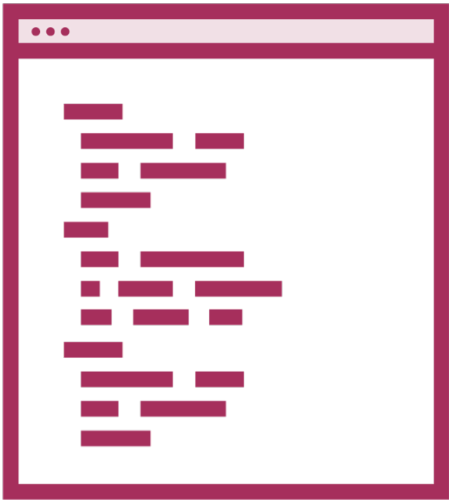
Adding the API project

Configuring the Startup class



Transitioning from View Services to MediatR





“Simple” controllers

- Heavy on code
- Validate incoming data using model binding
- Execute logic
- Create response type
- Return status code and response



Simple but Heavy Controller

```
public class CategoryController : Controller
{
    private readonly ICategoryRepository _categoryRepository;

    public CategoryController(ICategoryRepository categoryRepository)
    {
        _categoryRepository = categoryRepository;
    }

    public async Task<IActionResult> GetCategories()
    {
        var categories = await _categoryRepository.GetAll();

        var categoryViewModels = categories.Select(o => new CategoryViewModel() {Name = c.Name });

        return View(categoryViewModels);
    }
}
```





View services

- Logic is moved to separate class
- Called from controller
- Views service connects with business logic
- Returns DTO
- Results in more lightweight controllers



View Service

```
public class CategoryViewService : ICategoryViewService
{
    private readonly ICategoryRepository _categoryRepository;

    public async Task<IEnumerable<CategoryViewModel>> GetCategories()
    {
        var categories = await _categoryRepository.GetAll();

        var categoryViewModels =
            categories.Select(o => new CategoryViewModel() {Name = c.Name });

        return categoryViewModels;
    }
}
```



Using a View Service from the Controller

```
public class CategoryController : Controller
{
    private readonly ICategoryViewService _categoryViewService;

    public CategoryController(ICategoryViewService categoryViewService)
    {
        _categoryViewService = categoryViewService;
    }

    [HttpGet]
    public async Task<IActionResult> GetCategories()
    {
        var categoryViewModels = await _categoryViewService.GetCategories();
        return Ok(categoryViewModels);
    }
}
```





Using MediatR

- Specific type for query or command
- Controller sends via MediatR
- Handled in the Core project through RequestHandler
- Really lightweight controllers



Controllers Using MediatR

```
public class CategoryController : ControllerBase
{
    private readonly IMediator _mediator;

    public CategoryController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpGet("all", Name = "GetAllCategories")]
    public async Task<ActionResult<List<CategoryListVm>>> GetAllCategories()
    {
        var dtos = await _mediator.Send(new GetCategoriesListQuery());

        return Ok(dtos);
    }
}
```



Demo



Adding the correct packages

Creating the controllers



Deciding Which Objects to Return



What Should We Return?

List

Relevant properties
Wrapped in a `List<T>`
Typically a view model

Detail

All properties
View model
Nested DTO optional



Using a Response Type

```
public class BaseResponse
{
    public BaseResponse(string message, bool success)
    {
        Success = success;
        Message = message;
    }

    public bool Success { get; set; }

    public string Message { get; set; }

    public List<string> ValidationErrors { get; set; }
}
```



Demo



Exploring the options to return data

Using a specific response





“Hey, Mary here.

**Did I mention we’ll need to export the list
of events also to a CSV file?”**



Demo

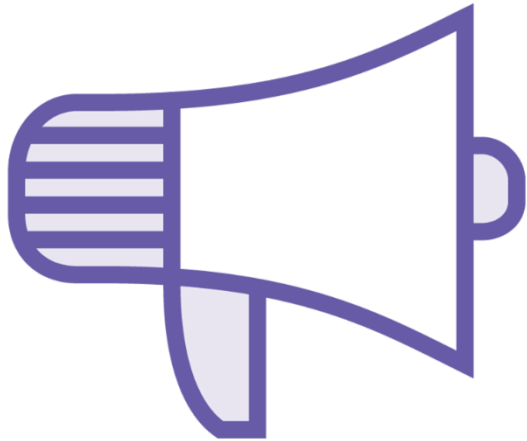


Implementing a new feature from the API
to the Core



Exposing the API Functionality Using Swagger

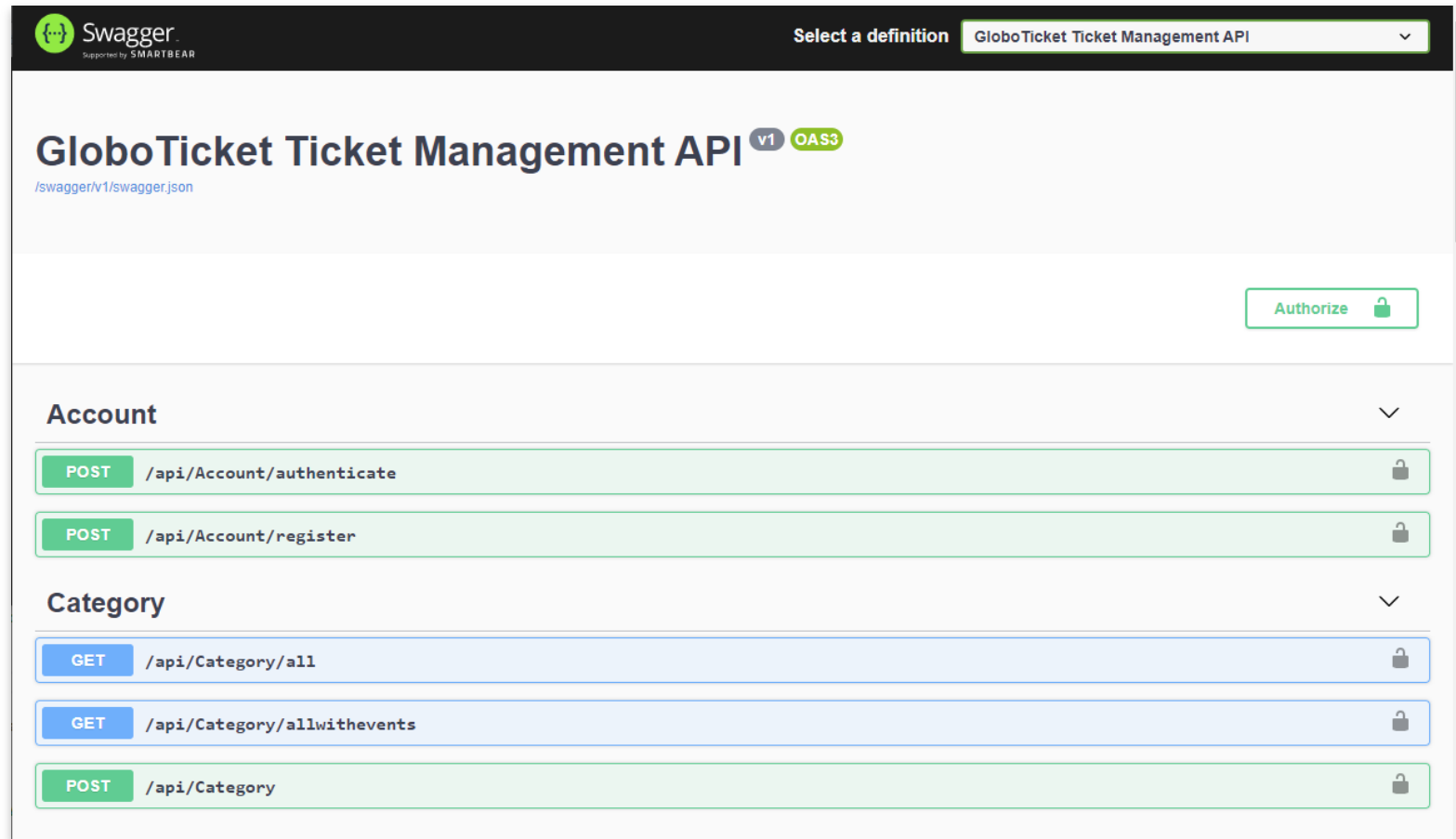




Adding Swagger

- API description
- Specification
 - JSON or YAML
- Tooling
 - Swashbuckle
 - NSwag (clients)

Exposing the Swagger API Contract



Required Packages

Swashbuckle
.AspNetCore

Swashbuckle
.AspNetCore
.Swagger

Swashbuckle
.AspNetCore
.SwaggerUI



Demo



Configuring Swagger and Swashbuckle



Summary



ASP.NET Core API is sending messages

Configured using Swagger

Adding a new functionality is easy





Up next:
Testing the architecture

