TALLINN UNIVERSITY OF TECHNOLOGY

IT College

Marko Linde 176292IADB

PRODROAD

Web Applications with C# (ICD0024)

Instructor: Andres Käver

Tallinn 2024

Tallinn 2024

Author declaration

The purpose of the project is to make a simple and user friendly, easily understandable and partly automated production planning tool. It is meant for small production companies and gives an opportunity to offer ten times cheaper alternative to many APP-s out there. Normal usable ready solutions with basic (no IoT support) usage start at 50-200 dollars per month, which is way too expensive for starting business and real small productions. Not mentioning the scale up of price by users and features.

Abbreviations and definitions

DTO	*Data transfer object*, object, that is meant for keeping data without functionality
C#	*C sharp*, third generation programming language
REST	*Representational state transfer,* application communication standard which separates client from the server.
MVC	*Model-View-Controller,* architectural pattern, that separates API to structural logical components. *Model* represents program logical component, *View* represents user component and controller the interface between user and server
API	*Application programming interface,* interface which helps the program to communicate to outside world
DAL	*Data Access Layer*
BLL	*Business Logic Layer*
JSON	*JavaScript Object Notation,* structure for data transfer
JWT	*JSON web token,*
UI	*User interface,*
DOM	*Document Object Model,* Html API
UX	*User experience,*
CRUD	*Create-Read-Update-Delete,*
SOLID	
UOW	
GIT	

Table of Contents

1.Assignment

APP where the program server component is written on C#. The frontend APP should end up eventually as mobile APP and as browser APP. The application should work in multiple canvas modes:

- Calendar mode is the view where user can make a roadmap for their teams, production lines and get a grasp of work collisions, workloads and rising problems in flow.
- Modular mode should give a view in different user created tasks. These tasks depend on company specific production flow (like assembly, packaging, production, sales, delivery and so on)
- Analytical mode is about giving feedback how planning and reality fit together. How different teams perform and what can be made better
- Resource mode should show stock of materials and when will they run out and give ordering predictions based on last arrivals from suppliers

Production Manager APP v0.1 and this project scope includes preliminary calendar, order and production management modes.

Future aspect includes mobile API for easier component utilization and notification management and reaction. API must have the possibility to synchronize data between several different IoT devices. Possible future solution should include export and import of warehouse component list and basic logistics.

2.Overview

ProdRoad APP v0.1 gives the production order, item management and workflow planning.

Server side DTO-s will be constructed from visual ERD-CASE diagram and database update and initialization will come through migration. Code first script files for database generation will be included at the end of this document.

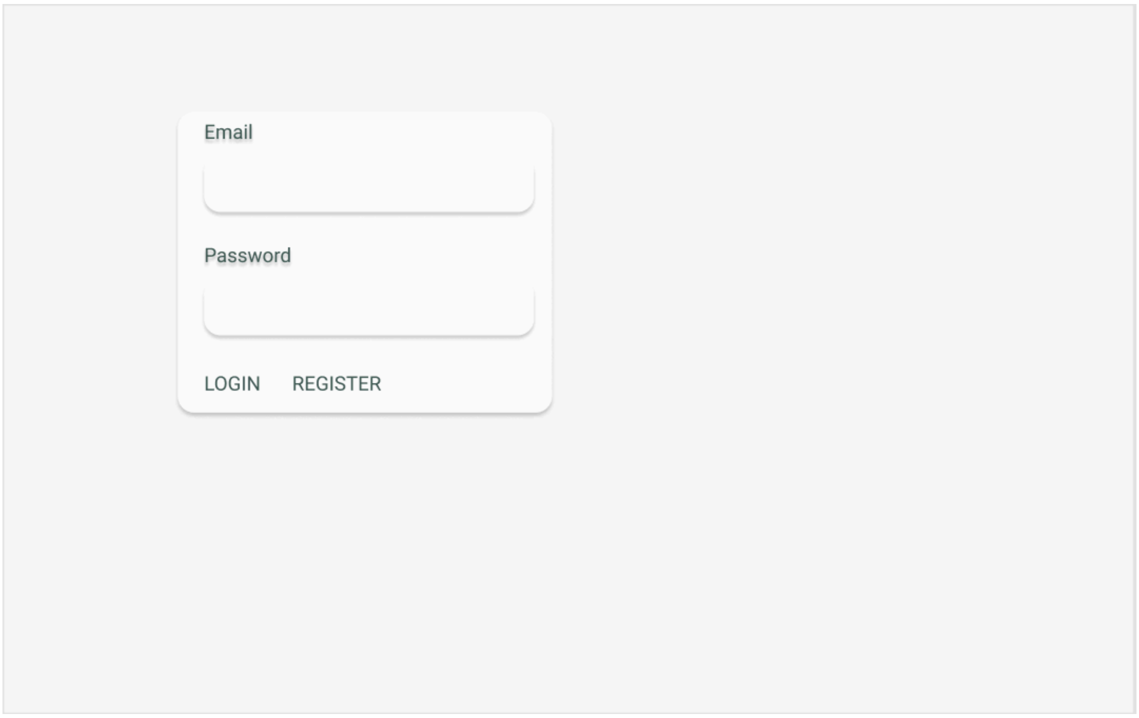
2.1.Client side business requirements

- program has to be written in C# language using .Net 8
- program has to follow clean SOLID architecture and MVC pattern
- program features have to be layered for easy change.
- Data will be stored using Postgres.
- Entity Framework will be used as wrapper over database and takes care of information mapping and virtual structure generation of DAL DTO-s
- DAL layer
- BLL layer
- API layer
- user creation and user activities basics are left for .Net individual authentication and identitycontext.
- User roles are : admin, manager and user
- Role groups are segregated between manager created Teams
- user client authorization will be with stateless JWT. JWT will get a secret which will expire with new key. JWT will also be added to every user request.

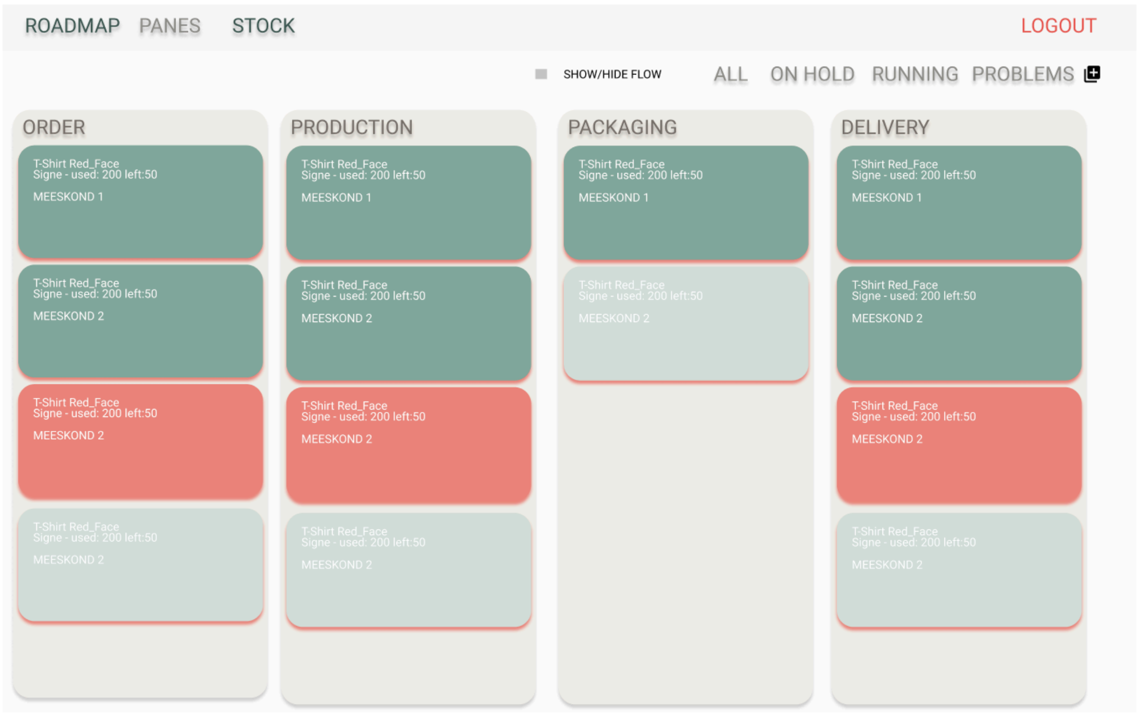
2.2.User interface

Main views made in Figma and interactive link:

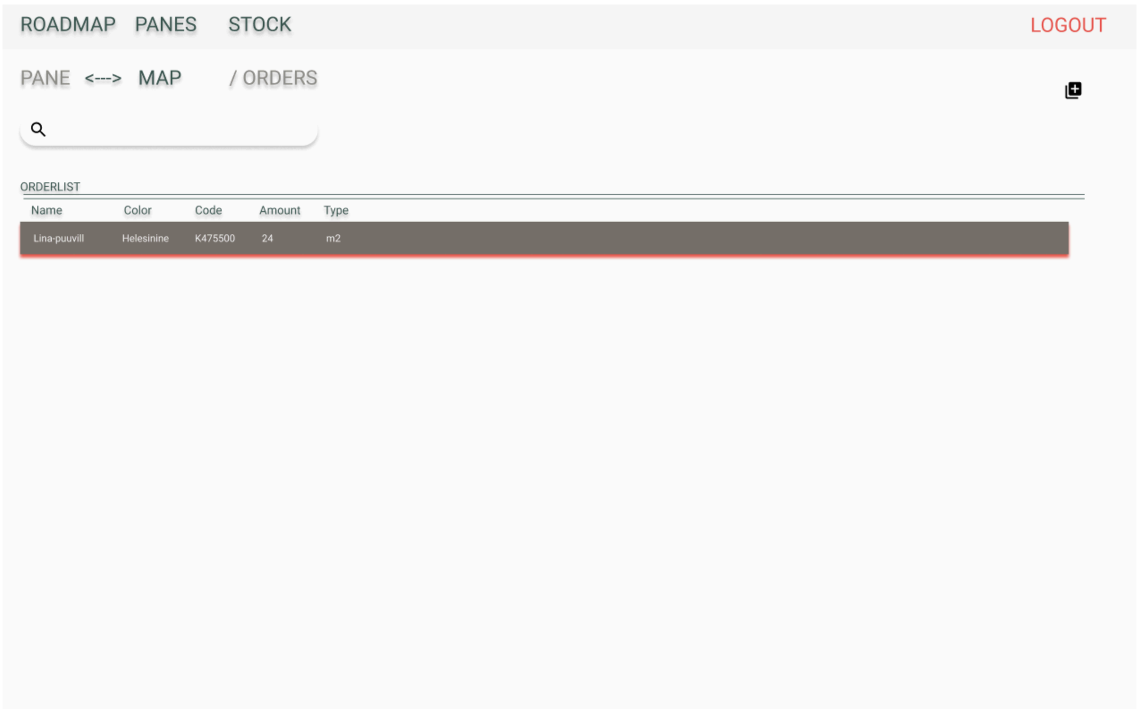
https://www.figma.com/proto/B3btV75evn32c4pV20fqPO/ProductionAPP?node-id=31%3A26&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=31%3A26&show-proto-sidebar=1



Drawing 1. Login screen

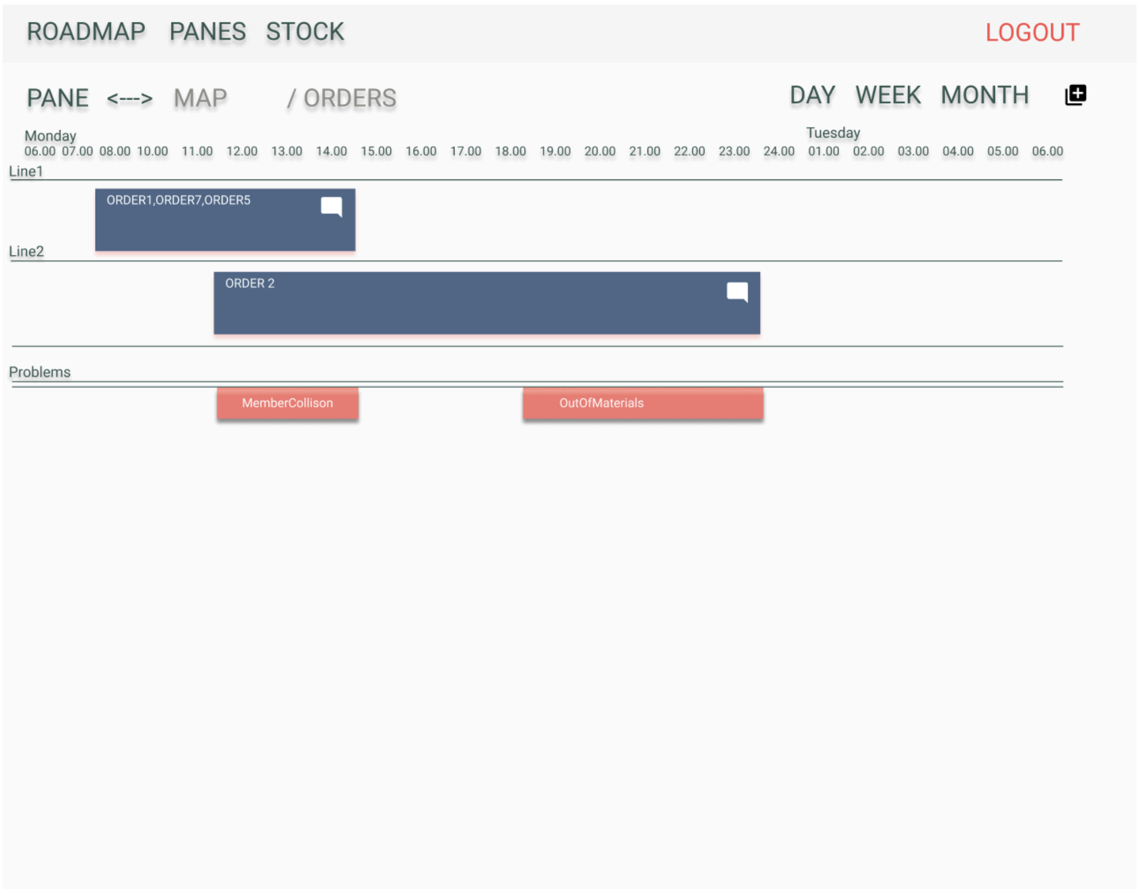


Drawing 2. Pane overview

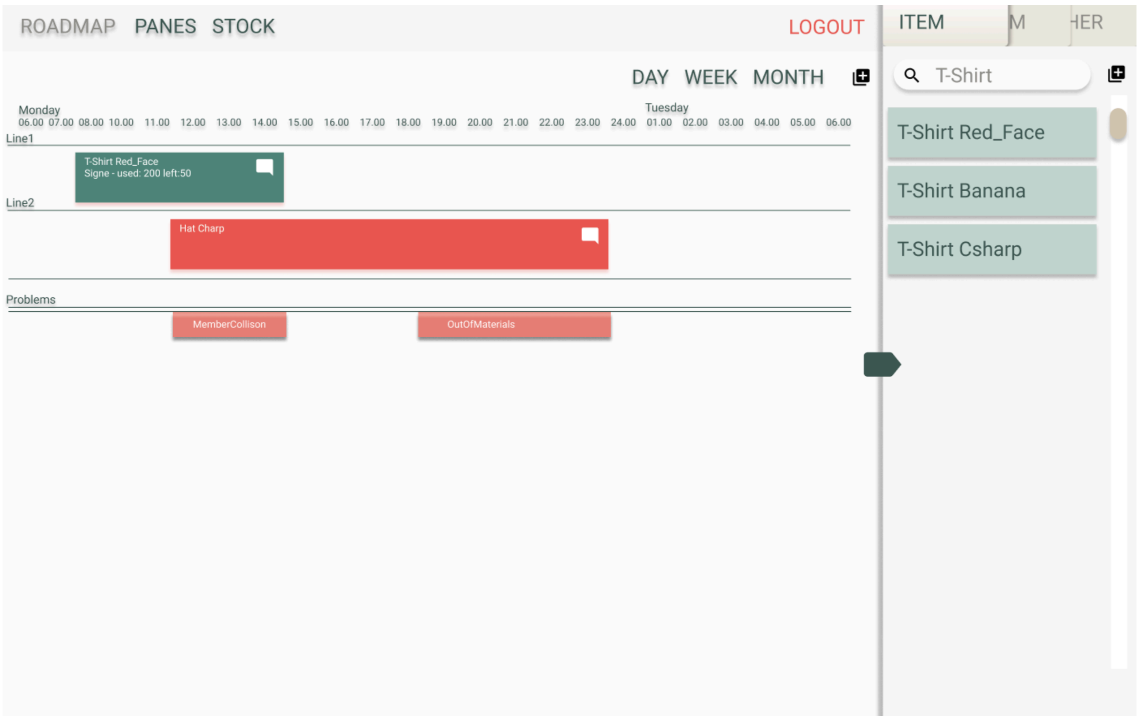


Name	Color	Code	Amount	Type
Lina-puuvill	Helesinine	K475500	24	m2

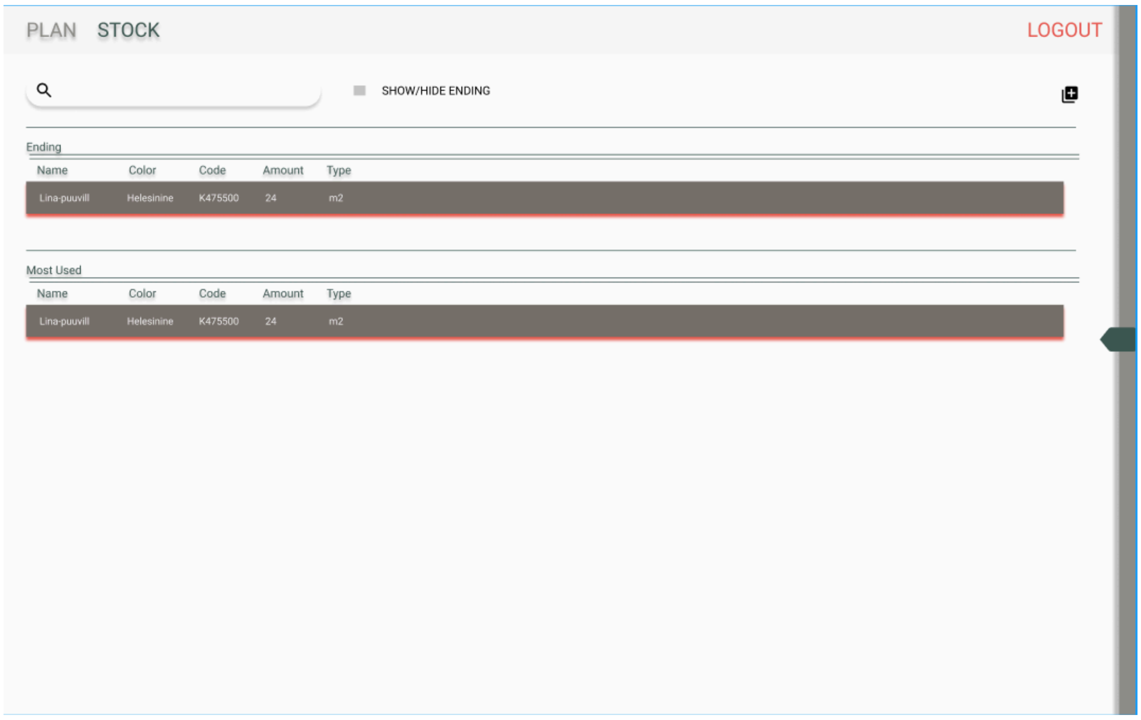
Drawing 3. Order list pane screen



Drawing 4. Order list map screen



Drawing 5. Production plan screen



Ending

Name	Color	Code	Amount	Type
Lina-puuvill	Helesinine	K475500	24	m2

Most Used

Name	Color	Code	Amount	Type
Lina-puuvill	Helesinine	K475500	24	m2

Drawing 6. Stock overview

- Server-client communication datatype will be JSON
- UI development for lesser mistakes will be held in typescript
- UI and some UX will be configured by Materialize library.

2.3.User access and roles

- ProdPlan APP has 4 different type of roles:
  - admin – for super acces of all operations
  - manager – for APP subscription and maintainer role
  - userManager – for specific Team realated operations
  - user – for view and message only
- Manager can access (CRUD) only its own or user related data assigned to it

- User can access (CRUD) its own and Manager related (mostly read, and some places CR)
- User can read Manager and userManager created data. CRUD only for messaging

2.4.UserManager functionality

- notifications CRUD
- fulfilling orders, marking extensions and problems

2.5.Datetime in database

All datetime values are saved in UTC format and are converted inside AppDbContext.

2.6.Data annotations – describe restrictions and allowed methods.

2.7.Used patterns

- Using repository pattern for encapsulating different logical layer for accessing data sources. Since different classes might have different data access logic, it is good to describe the repositories for more understandable data exchange and layer swapping. Gives also overall better understanding of API architecture.
- Using unit of work (instead of database context) for capsulating together all used repository classes for easier access. Dependency is injected into builder services as SCOPED (connection created by http request and closed after usage). UOW interface used in API controllers for data access.
- Factory pattern is used inside UOW class, where UOW interface should return either new repository or use already created one. Already created repositories are kept in dictionary

- Generics – are used for base elements.
    - This API allows to change the entity ID type (default is GUID)
    - Repository creation factory methods
- Overwrite methods capability for large amount of methods (virtual) -describe better

2.8.Translations

Translations are kept in Postgres database as JSONB format. This will reduce the need of multiple language support database tables for same entities and reduce complexity of reffering to multiple db tables.

Translations are accessed by URI specific culture mark:

- https://localhost:7222/api/Address?culture=et-EE

Et specifies base language, and -EE specifies the language regional usage.
Fallback culture is set to et-EE and API supports additional en-GB culture. Example en-GB or en-US where GB marks Great Britain and US United States languages.

Adding more languages needs modification of properties/appsettings.

Class property is referenced with property `[Column(TypeName = "jsonb")]`
and the datatype is changed to user defined LangStr :

- `public LangStr Head { get; set; } = new();.`

LangStr class handles from and to property conversions between different cultures.
JSONB (https://www.postgresql.org/docs/9.4/datatype-json.html) data structure is considered to be faster in execution, since it is stored as decomposed binary and it needs less reparsing. JSONB is slower at input, but removes property whitespace (less dataspace consumption) and supports indexing if needed. Additionally it does not allow duplicate keys, thus reducing error in JSONB setup.

Controllers (annex2) need additional help to convert in and output between string and JSONB datasets. This is achieved by entity specific DTO classes.

DELETE operation deletes entire row and culture deletion is not supported (out of scope)

Supported cultures can be added via controller PUT statement.

2.9.Testing

1) Postman testing cases:

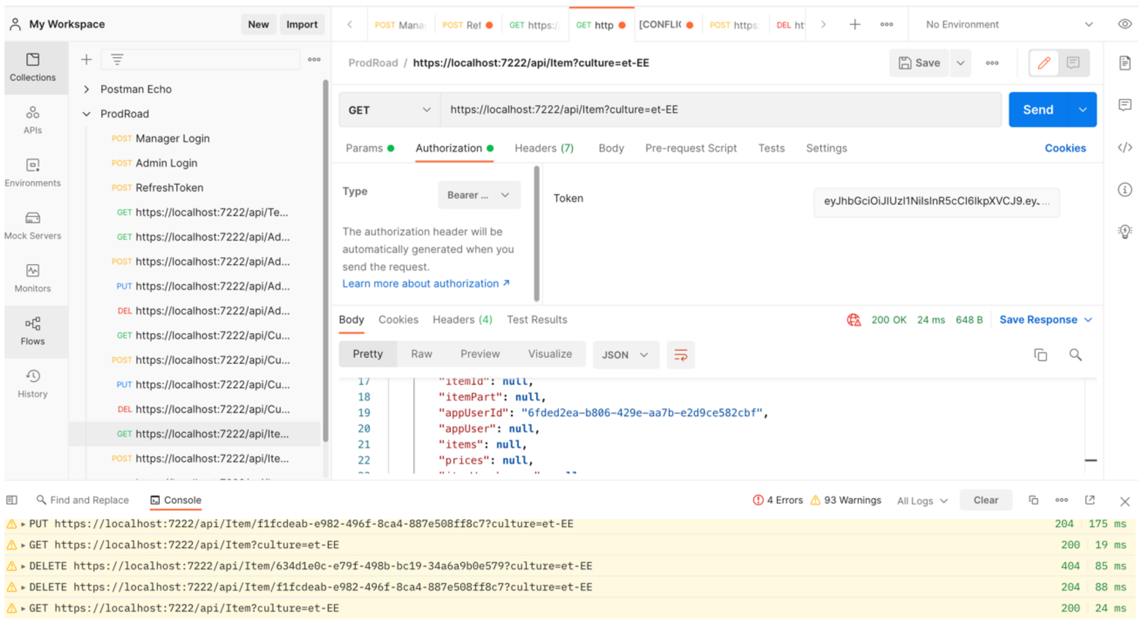
- GET https://localhost:7222/api/Address?culture=et-EE
- GET https://localhost:7222/api/Address/4cd15056-1b99-4528-9524-c4bd3a7c78d1?culture=et-EE
- PUT https://localhost:7222/api/Address/4cd15056-1b99-4528-9524-c4bd3a7c78d1?culture=ee-ET
- PUT https://localhost:7222/api/Address/4cd15056-1b99-4528-9524-c4bd3a7c78d1?culture=en-GB
- POST https://localhost:7222/api/Address?culture=ee-ET
- DELETE https://localhost:7222/api/Address/7dcf85e9-9c8e-49ec-a1e1-0b877eb6c8fd?culture=ee-ET

JSON:

```
{
    "appUserId": "9504bbb0-ab82-4af2-9ba5-f0ffb77ae23e",
    "appUser": null,
    "customerId": null,
    "customer": null,
    "country": "Eesti",
    "city": "Tartu",
    "street": null,
    "code": null,
    "phone": null,
    "email": null,
    "id": "4cd15056-1b99-4528-9524-c4bd3a7c78d1"
}
```

2.9.1.Testing with JWT (postman)

After testing JWT lifetime and access, it is feasible to make the token lifetime longer then 1 minute. Otherwise most of the testing time is spent on manual token refreshing, coping and pasting JWT data to CRUD operations


My Workspace
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Postman Echo
ProdRoad
POST Manager Login
POST Admin Login
POST RefreshToken
GET https://localhost:7222/api/Te...
GET https://localhost:7222/api/Ad...
POST https://localhost:7222/api/Ad...
PUT https://localhost:7222/api/Ad...
DEL https://localhost:7222/api/Ad...
GET https://localhost:7222/api/Cu...
POST https://localhost:7222/api/Cu...
PUT https://localhost:7222/api/Cu...
DEL https://localhost:7222/api/Cu...
GET https://localhost:7222/api/Ite...
POST https://localhost:7222/api/Ite...
No Environment
ProdRoad / https://localhost:7222/api/Item?culture=et-EE
Save
GET
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Bearer ...
Token
Learn more about authorization ↗
Body
Cookies
Headers (4)
Test Results
200 OK 24 ms 648 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
Find and Replace
Console
4 Errors
93 Warnings
All Logs
Clear

URL: https://localhost:7222/api/Item?culture=et-EE

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

The authorization header will be automatically generated when you send the request.

```
"itemId": null,
"itemPart": null,
"appUserId": "6fded2ea-b806-429e-aa7b-e2d9ce582cbf",
"appUser": null,
"items": null,
"prices": null,
```

```
⚠ ▸ PUT https://localhost:7222/api/Item/f1fcdeab-e982-496f-8ca4-887e508ff8c7?culture=et-EE        204   175 ms
⚠ ▸ GET https://localhost:7222/api/Item?culture=et-EE                                              200    19 ms
⚠ ▸ DELETE https://localhost:7222/api/Item/634d1e0c-e79f-498b-bc19-34a6a9b0e579?culture=et-EE     404    85 ms
⚠ ▸ DELETE https://localhost:7222/api/Item/f1fcdeab-e982-496f-8ca4-887e508ff8c7?culture=et-EE     204    88 ms
⚠ ▸ GET https://localhost:7222/api/Item?culture=et-EE                                              200    24 ms
```

3.Architecture – describe catalogue setup and API layering

3.1.User detection and role management.

- MVC controller roles are set on top of controller.(example Teams controller)
- Users are detected inside repository classes, where data requests are overloaded.
- User ID is not sent to MVC view.

3.2.JWT

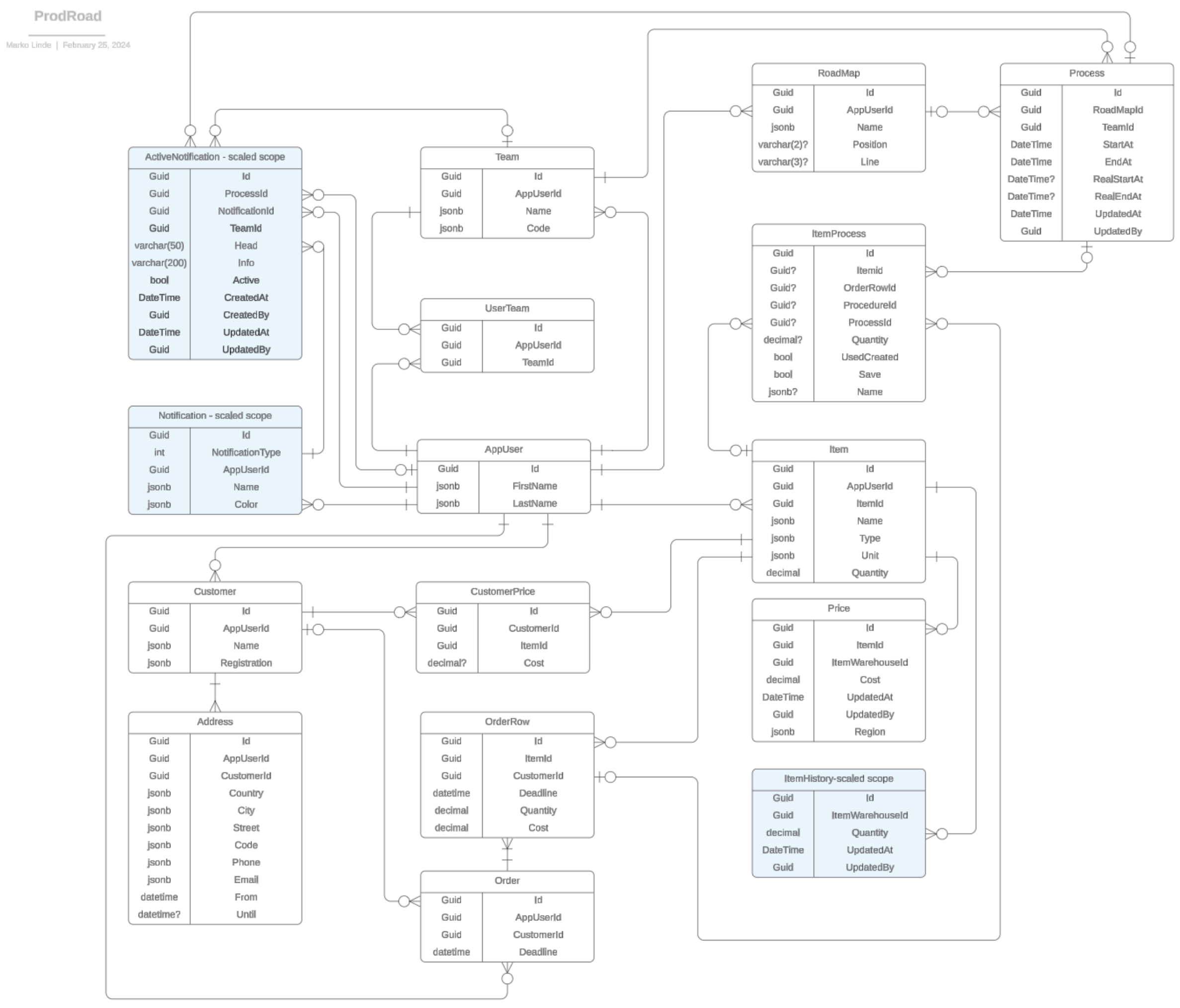
Using JWT for keeping user specific claims and authentication data. Is needed for keeping alive user logon and stateless session.

Using two type of tokens:

- JWT – expires in one minute for better security and hinders token snatching. But the token is alive for more then 1 minute to mitigate request-response collision risks.
- Refresh TOKEN – GUID string, which is alive for 7 days and helps to refresh JWT for data operations

Both tokens are refreshed every time user needs JWT for data access and user must log on with each device separately. Each device gets its own tokens. Refresh-token list is kept under user data.

Annex 1 – DB Model



Annex 2 – REST Address controller

```
[Route("api/[controller]")]
[ApiController]
public class AddressController : ControllerBase
{
    private readonly AppDbContext _context;

    public AddressController(AppDbContext context)
    {
        _context = context;
    }

    // GET: api/Address
    [HttpGet]
    public async Task<ActionResult<IEnumerable<AddressDTO>>>
GetAddresses()
    {
        var address = (await _context.Addresses
                .ToListAsync())
                .Select(ad => new AddressDTO()
                {
                    Id = ad.Id,
                    AppUserId = ad.AppUserId,
                    CustomerId = ad.CustomerId,
                    Country =  ad.Country,
                    City = ad.City,
                    Street = ad.Street,
                    Phone = ad.Phone,
                    Code = ad.Code,
                    Email = ad.Email
                })
                .ToList();
        return address;
    }

    // GET: api/Address/5
    [HttpGet("{id}")]
    public async Task<ActionResult<AddressDTO>> GetAddress(Guid id)
    {
        var dbAddress = await _context.Addresses.FindAsync(id);

        if (dbAddress == null)
        {
            return NotFound();
        }

        var address = new AddressDTO()
        {
            Id = dbAddress.Id,
            AppUserId = dbAddress.AppUserId,
            CustomerId = dbAddress.CustomerId,
            Country =  dbAddress.Country,
            City = dbAddress.City,
            Street = dbAddress.Street,
            Phone = dbAddress.Phone,
            Code = dbAddress.Code,
            Email = dbAddress.Email
        };
        return address;
    }

    // PUT: api/Address/5
    // To protect from overposting attacks, see https://
go.microsoft.com/fwlink/?linkid=2123754
    [HttpPut("{id}")]
    public async Task<IActionResult> PutAddress(Guid id, AddressDTO
address)
    {
        if (id != address.Id)
        {
            return BadRequest();
        }

        var dbAddress = await _context.Addresses.FindAsync(id);

        if (dbAddress == null) {return NotFound();}

        dbAddress.Country!.SetTranslation(address.Country!);
        dbAddress.City!.SetTranslation(address.City!);
        dbAddress.Street!.SetTranslation(address.Street!);
        dbAddress.Code!.SetTranslation(address.Code!);
        dbAddress.Phone!.SetTranslation(address.Phone!);
        dbAddress.Email!.SetTranslation(address.Email!);

        _context.Entry(dbAddress).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!AddressExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // POST: api/Address
    // To protect from overposting attacks, see https://
go.microsoft.com/fwlink/?linkid=2123754
    [HttpPost]
    public async Task<ActionResult<AddressDTO>> PostAddress(AddressDTO
address)
    {

        var dbAddress = new Address()
        {
            AppUserId = address.AppUserId,
            CustomerId = address.CustomerId
        };

        dbAddress.Country!.SetTranslation(address.Country!);
        dbAddress.City!.SetTranslation(address.City!);
        dbAddress.Street!.SetTranslation(address.Street!);
        dbAddress.Code!.SetTranslation(address.Code!);
        dbAddress.Phone!.SetTranslation(address.Phone!);
        dbAddress.Email!.SetTranslation(address.Email!);

        _context.Addresses.Add(dbAddress);

        await _context.SaveChangesAsync();

        return CreatedAtAction("GetAddress", new { id = address.Id },
address);
    }

    // DELETE: api/Address/5
    //TODO: implement culture deletion
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteAddress(Guid id)
    {
        var address = await _context.Addresses.FindAsync(id);
        if (address == null)
        {
            return NotFound();
        }

        _context.Addresses.Remove(address);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool AddressExists(Guid id)
    {
        return _context.Addresses.Any(e => e.Id == id);
    }
}
```