**Electronic-Shop**

**Szikszai Csaba**

**Grupa 30234**

# **Prezentare Generală**

Proiectul de față reprezintă o aplicație web care oferă posibilităti de achiziționare a produselor electronice. Acest sistem folosește o bază de date pentru stocarea datelor necesare utilizatorilor (nume, adresa, email, număr de telefon, poză de profil), produselor (nume, preț, stoc, poză, reducere) și comenzilor (timpul achiziționării, utilizatorul, lista de produse).

Prin comparație cu alte produse software aș putea spune că aplicația web Electronic-Shop se aseamănă într-un anumit procent cu aplicatia web Emag. Interfața utilizatorului va fi diferită de cea a aplicației menționate anterior, iar funcționalitățile vor fi mult mai restrânse.

# **Tehnologie IT**

În realizarea produsului software am folosit programarea orientată pe obiecte, folosindu-ne de limbajul de programare **C#** și framework-ul **.NET 6** pe partea de Back-End și framework-ul **Angular** pe Front-End.

Pentru a ușura munca programatorilor și pentru a economisi mai mult timp, **Microsoft** pune la dispoziție o serie de documente de calitate, pe diferite categorii de subiecte unde se pot găsi informații necesare folosrii framework-ului **.NET 6** pentru crearea unei **API.**

De asemenea **Angular** pune la dispoziție o serie de documente de calitate, pe diferite categorii de subiecte.

# **Architectura și Designul Sistemului**

## **Diagrama Flow-Chart**

Diagram

Description automatically generatedEste o imagnie a pașilor separați a unui proces, redat în ordine secvențială. În cazul de fată am descris succesiunea de pași necesară achiziționării unui produs.

## **Cazuri de utilizare**

**Diagrama cazurilor de utilizare a utilizatorului**

Diagram

Description automatically generatedUtilizatorul se poate înregistra în sistem, se poate autentifica, vizualiza produse, caută produse după nume, daca este autentificat poate să adaugă produse la favorite sau coș, poate să plaseaza o comandă, să modifice detaliile profilului.

**Diagrama cazurilor de utilizare a administratorului**

Diagram

Description automatically generatedAdministratorul poate adăuga produse, modifica produse, vizualiza toți utilizatorii, poate căuta utilizatori după nume sau e-mail și șterge un utilizator.

## **Diagrama de clase**

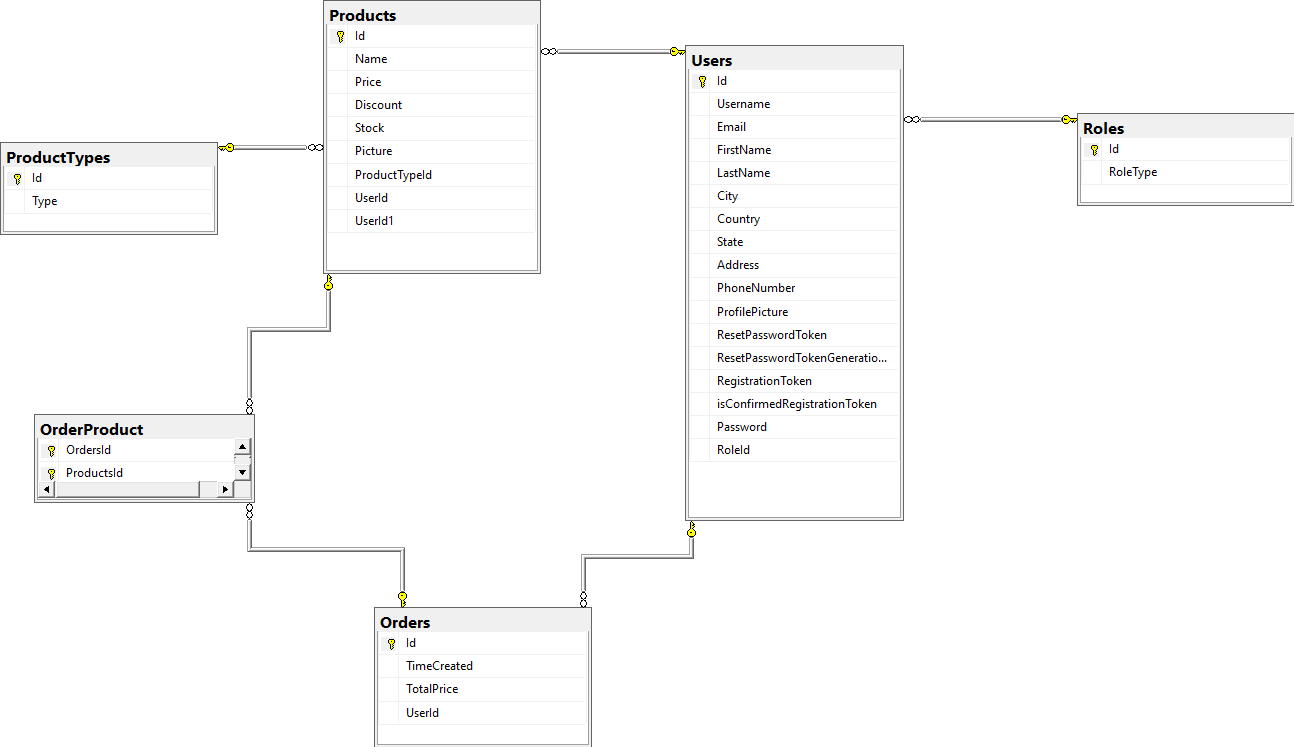
Este un tip de diagramă de structură statică care descrie structura sistemului arătând clasele sistemului, atributele lor, capabilitățile și relațiile dintre ele.

A picture containing graphical user interface

Description automatically generated

## **Diagrama bazei de date**

Diagrama bazei de date arată grafic structura bazei de date. Folosind diagrame de baze de date putem crea și modifica tabele, coloane, relații și chei. În plus, putem modifica indexuri și constrângeri.



## **Diagrama de desfășurare**

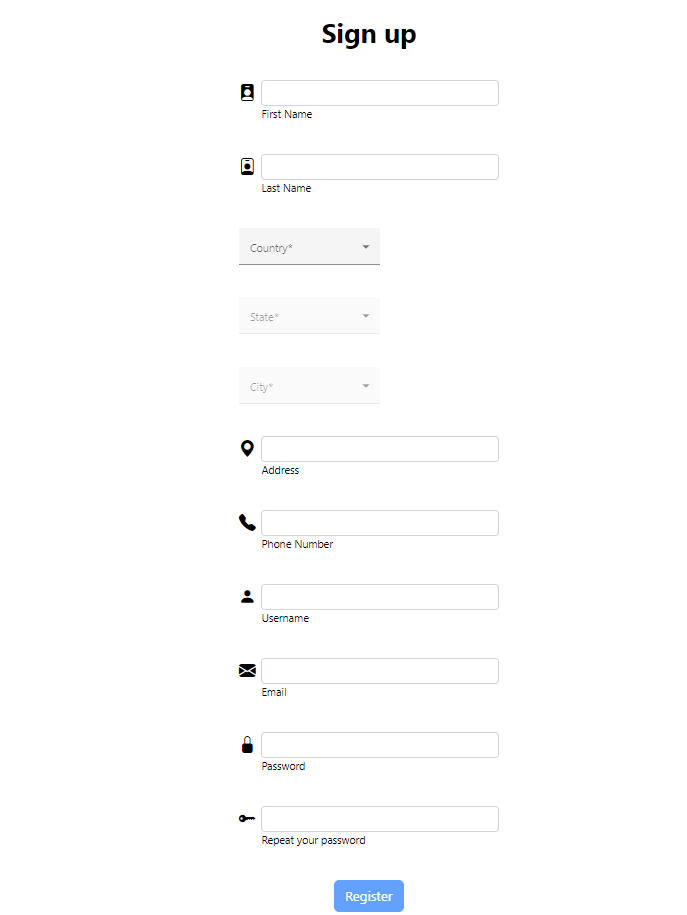
Diagramele de desfășurare sunt utilizate pentru a vizualiza topologia componentelor fizice ale unui sistem, unde sunt implementate componentle software.

Diagramele de desfășurare sunt utilizate pnetru a descri vizualizarea statică a implementării unui sistem și sunt compuse din noduri și relațiile lor.

Text

Description automatically generated with low confidence

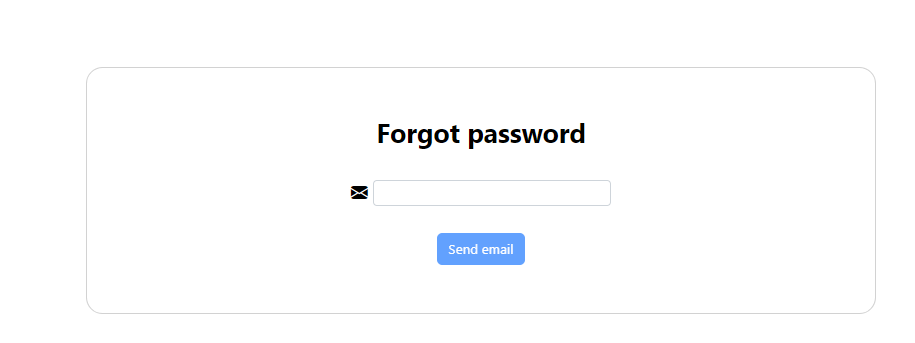
## **Interfața utilizatorului**

Graphical user interface, application

Description automatically generated

Graphical user interface, application

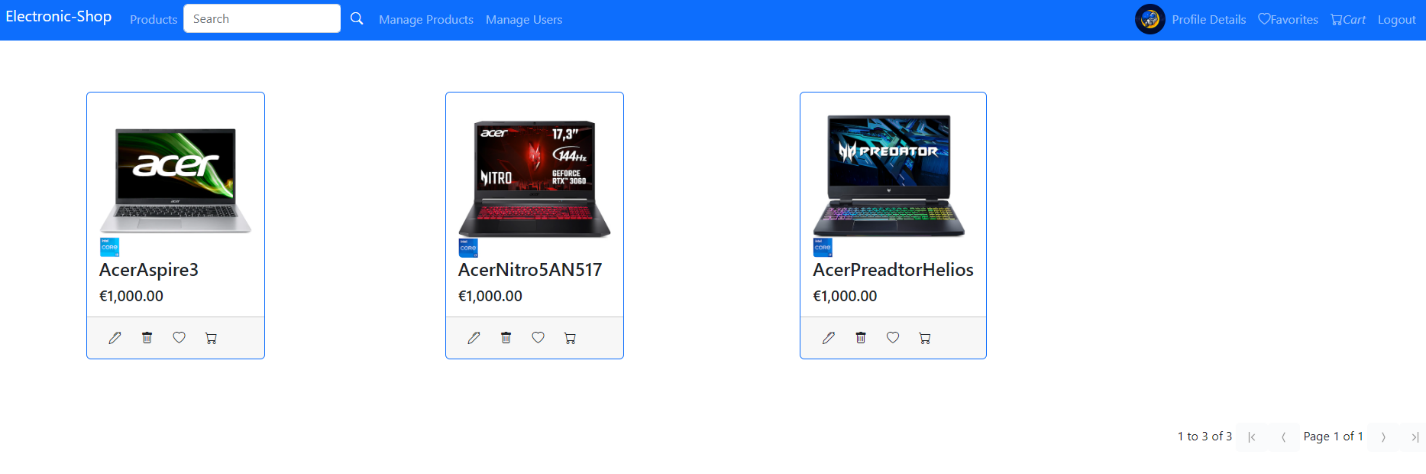
Description automatically generatedGraphical user interface, application

Description automatically generated

A picture containing graphical user interface

Description automatically generated

## **Interfața administratorului**



Graphical user interface, website

Description automatically generated

A screenshot of a computer

Description automatically generated

# Graphical user interface, text, application, email Description automatically generated**Tratarea errorilor**

Pentru tratarea errorilor am implementat un Error Handler Middleware, care preia mesajele din excepții și trece la Front-End unde un error-handler-service preia, gestionează datele și afișează mesajele pentru utilizatori.

Diagram, schematic

Description automatically generated

# **Autentificare și autorizare**

**Registrare**

Un utilizator poate să înregistreze navigând la pagina de “Registration” și introducând datele personale și apăsând butonul de Register. Pentru a se putea autentifica, utilizatorul trebuie mai întâi să verifice e-mailul făcând click pe linkul primit în e-mail.

Servicile de email au fost implementate cu API-ul de SendGrid.

**Autentificare**

Pentru a se autentifica, utilizatorul trebuie să introducă numele de utilizator și parola cu care s-a înregistrat. La autentificare se codifică rolul, id-ul și numele utilizatorului într-un **JSON Web Token** care se salvează în **Local Storage**.

**Autorizare**

Autorizarea se face pe rolul utilizatorului. La fiecare cerere de http, se preia rolul din **JSON Web Token** și verifică dacă utilizatorul are autorizație pentru metodă.

# **Bibliografie**

Documentație Angular: <https://angular.io/docs>

Bootstrap: <https://ng-bootstrap.github.io/#/getting-started>

Angular Material: <https://material.angular.io>

AG Grid: <https://ag-grid.com/angular-data-grid/>

Documentație .Net: <https://learn.microsoft.com/en-us/dotnet/>

SendGrid: <https://sendgrid.com>

JSON Web Token: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/jwt-authn?view=aspnetcore-7.0&tabs=windows>

**Generare Json Web Token :**

public async Task<string> GenerateJwtTokenAsync(GenerateJwtDto generateJwtDto) {

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_configuration.GetSection("Jwt:Key").Value);

var tokenDescriptor = new SecurityTokenDescriptor {

Subject = generateJwtDto.Claims,

Expires = generateJwtDto.ExpirationDate,

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

public async Task<string> GetAuthentificationJwtAsync(GetDetailsForLoginDto user) {

var claimsIdentity = new ClaimsIdentity(new[] {

new Claim("id", user.Id.ToString()),

new Claim(ClaimTypes.Name, user.Username),

new Claim(ClaimTypes.Role, user.Role.ToString()) });

var expires = DateTime.UtcNow.AddHours(6);

var generateJwtDto = new GenerateJwtDto {

Claims = claimsIdentity,

ExpirationDate = expires

};

return await GenerateJwtTokenAsync(generateJwtDto);

}

**Trimitere Email:**

public string CreateForgotPasswordMessageBody(SendForgotPasswordEmailDto sendForgotPasswordEmailDto) {

var bodyBuilder =

$"You have {AppConstants.TOKEN\_VALIDATION\_TIME} minutes to click the link and reset your password " +

$"<a href={sendForgotPasswordEmailDto.Uri}>link</a>";

return bodyBuilder;

}

public async Task SendForgotPasswordEmailAsync(SendForgotPasswordEmailDto sendForgotPasswordEmailDto) {

var apiKey = \_config["SendGrid:ApiKey"];

var client = new SendGridClient(apiKey);

var from = new EmailAddress(\_config["SendGrid:Email"]);

var subject = $"ElectricShop Forgot Password";

var to = new EmailAddress(sendForgotPasswordEmailDto.Email);

var plainTextContent = "";

sendForgotPasswordEmailDto.Uri = Path.Combine(AppConstants.FE\_APP\_URL, sendForgotPasswordEmailDto.Token);

var htmlContent = CreateForgotPasswordMessageBody(sendForgotPasswordEmailDto);

var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);

await client.SendEmailAsync(msg);

}

public string CreateVerificationMessageBody(SendVerificationEmailDto sendVerificationEmailDto) {

var bodyBuilder =

$"You have {AppConstants.VERIFICATION\_TOKEN\_VALIDATION\_TIME} hours to click the link before it expires " +

$"<a href={sendVerificationEmailDto.Uri}>link</a>";

return bodyBuilder;

}

public async Task SendVerificationEmail(SendVerificationEmailDto sendVerificationEmailDto) {

var apiKey = \_config["SendGrid:ApiKey"];

var client = new SendGridClient(apiKey);

var from = new EmailAddress(\_config["SendGrid:Email"]);

var subject = $"ElectricShop verify email";

var to = new EmailAddress(sendVerificationEmailDto.Email);

var plainTextContent = "";

sendVerificationEmailDto.Uri = Path.Combine(AppConstants.FE\_APP\_CR\_URL, sendVerificationEmailDto.Token);

var htmlContent = CreateVerificationMessageBody(sendVerificationEmailDto);

var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);

await client.SendEmailAsync(msg);

}

**Error Handler MiddlewareL:**

public class ErrorHandlerMiddleware {

private readonly RequestDelegate \_next;

public ErrorHandlerMiddleware(RequestDelegate next) {

\_next = next;

}

public async Task InvokeAsync(HttpContext context) {

try {

await \_next(context);

}

catch (Exception exception) {

var response = context.Response;

response.ContentType = "application/json";

switch (exception) {

case ModelNotFoundException e:

response.StatusCode = (int)HttpStatusCode.NotFound;

break;

case AuthenticationException e:

response.StatusCode = (int)HttpStatusCode.BadRequest;

break;

case PasswordException e:

response.StatusCode = (int)HttpStatusCode.Conflict;

break;

case AlreadyExistsException e:

response.StatusCode = (int)HttpStatusCode.Conflict;

break;

default:

response.StatusCode = (int)HttpStatusCode.InternalServerError;

break;

}

var result = JsonSerializer.Serialize(new { message = exception?.Message });

await response.WriteAsync(result);

}

}

}

**Program.cs:**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<DataContext>(options => {

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));

});

builder.Services.RegisterServices();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(options => {

options.AddSecurityDefinition("oauth2", new OpenApiSecurityScheme {

Description = "Standard Authorization header using the Bearer scheme",

In = ParameterLocation.Header,

Name = "Authorization",

Type = SecuritySchemeType.ApiKey

});

options.OperationFilter<SecurityRequirementsOperationFilter>();

});

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options => {

options.TokenValidationParameters = new TokenValidationParameters {

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8

.GetBytes(builder.Configuration.GetSection("Jwt:Key").Value)),

ValidateIssuer = false,

ValidateAudience = false

};

});

var app = builder.Build();

if (app.Environment.IsDevelopment()) {

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseCors(x => x

.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader());

app.UseStaticFiles(new StaticFileOptions {

FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).FullName, AppConstants.RESOURCE\_FOLDER, @"Resources")),

RequestPath = new PathString("/Resources")

});

app.UseCors("NgOrigins");

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.UseMiddleware<ErrorHandlerMiddleware>();

app.MapControllers();

app.Run();