

INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA
ENGENHARIA EM SISTEMAS INFORMÁTICOS

Relatório Integração de Sistemas de Informação

No. 23502 – Manuel Fernandes

Índice

Relatório Integração de Sistemas de Informação	1
Índice de Figuras.....	2
Introdução.....	4
Base de Dados.....	5
API	6
RESTful API.....	7
<i>Appsettings.json</i>	7
Program.cs	8
Services	9
Models	11
Controllers	12
JWT Authentication	16
Integração Externa – <i>Stripe</i>	19
SOAP API.....	21
Models	22
Service	24
Testes	27
Insomnia	27
Cloud.....	30
Conclusão	31
Referências.....	32

Índice de Figuras

Figura 1 - Base de Dados	5
Figura 2 - Estrutura da API.....	6
Figura 3 - appsettings.json	7
Figura 4 - Stripe Program.cs	8
Figura 5 - Swagger Documentation Program.cs.....	8
Figura 6 - JWT Token Program.cs.....	8
Figura 7 – JwtServices.....	9
Figura 8 – UserService	10
Figura 9 – UserModel.....	11

Figura 10 - UserController (GetAll)	12
Figura 11 - UserController (Post)	13
Figura 12 - UserController (Put)	14
Figura 13 - UserController (Delete)	15
Figura 14 - JWT Login Implementation	16
Figura 15 - Authorize Request	16
Figura 16 - Login Test	17
Figura 17 - Authorized User teste	18
Figura 18 - Controller Stripe	19
Figura 19 - Controller Stripe (1)	19
Figura 20 - SOAP API Estrutura	21
Figura 21 - SOAP PaymentModel	22
Figura 22 - SOAP PaymentResponseModel	22
Figura 23 - SOAP PaymentResponseModel	23
Figura 24 - SOAP PaymentStatusResponse	23
Figura 25 - PaymentServiceWS	24
Figura 26 - PaymentServiceWS(1)	24
Figura 27 - SOAP MakePayment	25
Figura 28 - SOAP RefundPayment	26
Figura 29 - Insomnia Variáveis	27
Figura 30 - Insomnia Estrutura	28
Figura 31 - Insomnia Test	29
Figura 32 - Azure Painel	30
Figura 33 - Azure SQL Database	30

Introdução

Este projeto consiste no desenvolvimento de uma aplicação de loja online, com o objetivo de permitir a gestão de produtos, utilizadores, encomendas e pagamentos. Para isso, foi criada uma *API* utilizando duas abordagens: *RESTful* para as funcionalidades principais e *SOAP* para o processamento de pagamentos, garantindo flexibilidade e integração com sistemas externos.

A autenticação dos utilizadores foi realizada com *tokens JWT*, assegurando a segurança na comunicação. A aplicação também inclui testes automatizados para garantir o funcionamento correto de suas funcionalidades e foi hospedada na plataforma *Azure* para garantir escalabilidade e disponibilidade.

Além disso, foi integrada a *API* do *Stripe* para o processamento de pagamentos, e a documentação da *API* foi realizada utilizando o *Open API Swagger*, facilitando a compreensão e utilização das funcionalidades da aplicação.

Base de Dados

A base de dados para este projeto foi feita em SQL, diretamente no *Microsoft SQL Server Management Studio* como está evidente na **Figura 1**. Apesar da base de dados ter sido também feita com recurso ao **Microsoft Azure**, infelizmente e acidentalmente, os créditos da mesma expiraram e não foi possível contar com o *deploy* quer seja da *api* ou da base de dados na *cloud*.

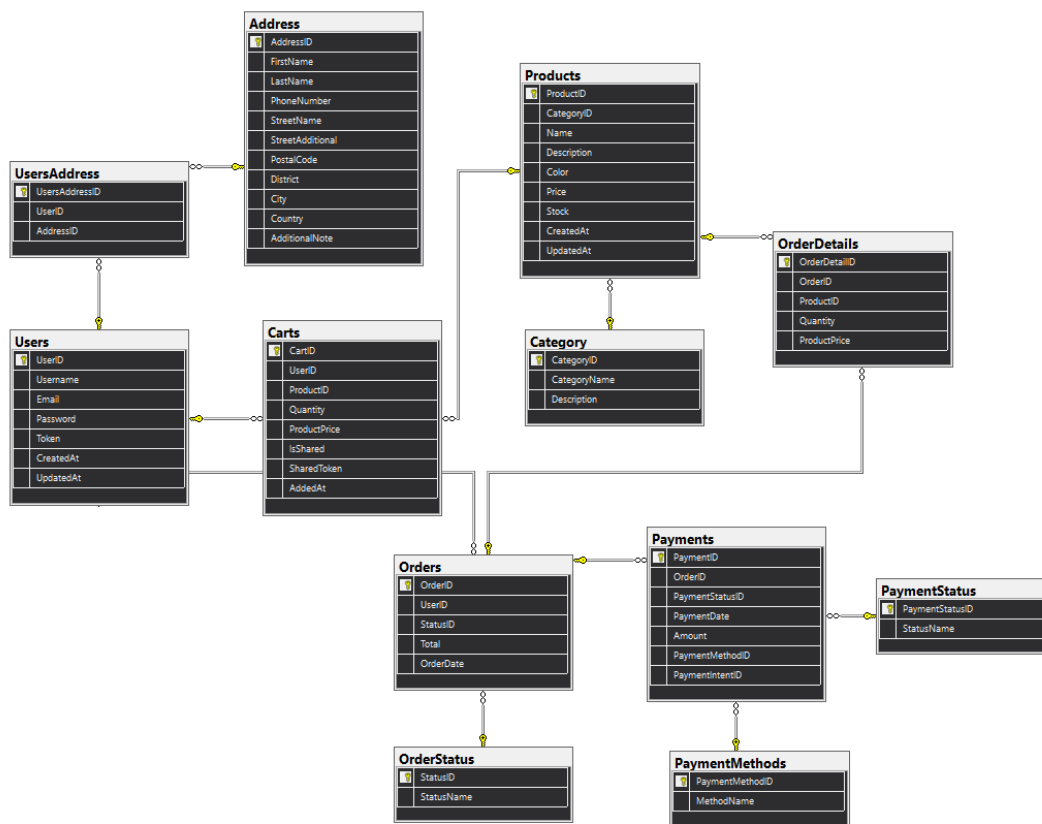


Figura 1 - Base de Dados

API

Para este projeto, foram desenvolvidas *APIs* em *RESTful* e *SOAP*. Sendo que o foco principal foi na vertente *RESTful*, já que apenas a parte dos Pagamentos foram feitos em *SOAP*.

Na Figura 2, podemos observar a estrutura das pastas e ficheiros da solução da *api*. Dentro da solução podemos encontrar a *RESTful API* e a *SOAP API* devidamente separadas em projetos diferentes na mesma solução

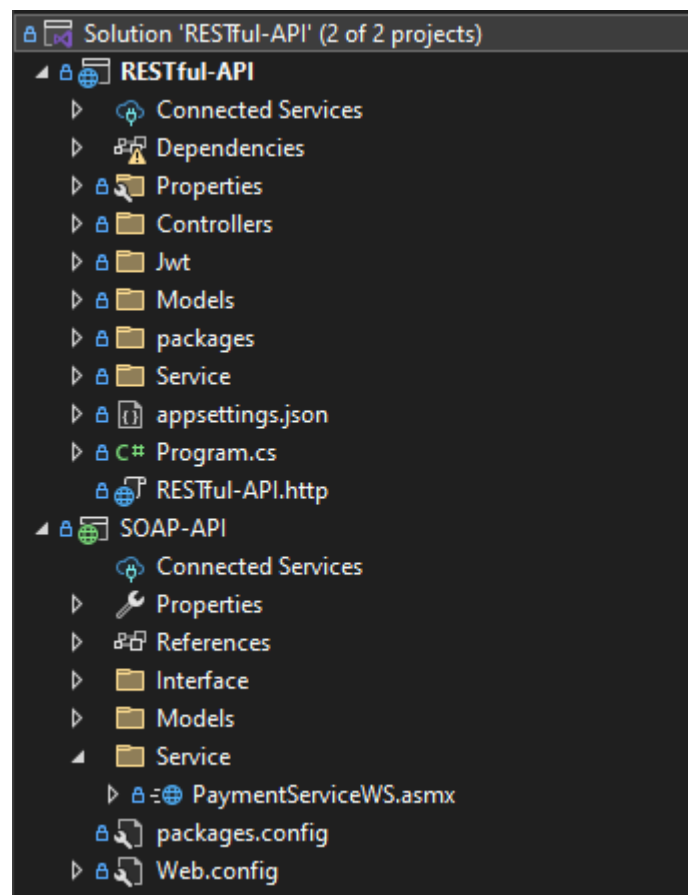


Figura 2 - Estrutura da API

De seguida, damos início à exploração destas *API*.

RESTful API

Foi desenvolvida uma *API RESTful* que cobre todas as operações necessárias para a gestão da loja online, incluindo utilizadores, produtos, encomendas, carrinhos e pagamentos.

- Estrutura *MVC (Model-View-Controller)*.
- Uso do *Microsoft.Data.SqlClient* para interações com a base de dados.
- Conformidade com os princípios *RESTful*.
- Utilização de Tokens *JWT*

Esta *RESTful API* conta com *Controllers*, *Models*, *Services* e uma pasta *Jwt* que contém as *settings* dos Tokens *JWT*.

Appsettings.json

No ficheiro *appsettings.json*, através da Figura 3, podemos encontrar várias *connection strings* que foram utilizadas ao longo do projeto. Sendo as mais importantes, a “*Home PC Server*” e “*Azure SQL Server*”. Como referido anteriormente, infelizmente, o servidor da plataforma *Azure* não se encontra disponível pois os créditos da mesma expiraram.

```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "ConnectionStrings": {
9     // LocalServer Laptop
10    // "ISIWebAPI": "Data Source=DESKTOP-6CDU801;Initial Catalog=ISI_API;Integrated Security=True;Connect Timeout=30;Encrypt=True;Trust Server Certificate=True;Application Intent=ReadWrite;Multi Subnet Failover=False"
11
12    // Home PC Server
13    "ISIWebAPI": "Data Source=192.168.58.58,1433;Database=ISI_API;MultipleActiveResultSets=True;User Id=grupoll;Password=grupoll;TrustServerCertificate=True;"
14
15    // Azure SQL Server
16    // "ISIWebAPI": "Server=tcp:isilapi.database.windows.net,1433;Initial Catalog=ISI_API;Persist Security Info=False;User ID=manu;Password=*****;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connecti
17  },
18  "AllowedHosts": "*",
19  "Jwt": {
20    "Key": "njgdsB342589mgkdf$E$EgfdEGDFSD",
21    "Issuer": "ISI",
22    "Audience": "Customer",
23    "TokenExpirationMinutes": 5
24  },
25  "Stripe": {
26    "SecretKey": "sk_test_51QbK1e0TRaLngmaZuyw7e787jkl1Uz2FHoIRpIOnMmR4ly6xcF861owR1TQzC31RgGUvns1RWt1kh7uocAM2B889Tnzdsyd",
27    "PublishableKey": "pk_test_51QbK1e0TRaLngmaAayg37gwurM0rLkFNPONLUtvJHP3H3gEr90h21GuF16aSYhbQyFwqypP1b4UJW2Nzvsqg568zgfzRmDS"
28  }
29 }
```

Figura 3 - appsettings.json

Para além das ligações ao servidor, temos também os parâmetros necessários para a integração com o *Stripe* e com os Tokens *JWT*.

Program.cs

No ficheiro base, *program.cs*, de onde existem várias configurações necessárias para o funcionamento da API, podemos evidenciar as configurações da integração externa com o *Stripe*, *Swagger OpenAPI Documentation* e *Tokens JWT*. Os mesmos estão presentes nas Figuras 4, 5 e 6, respetivamente.

```
19 // Stripe Payment
20 var stripeKey = builder.Configuration["Stripe:SecretKey"]; // Add this to appsettings.json
21 if (string.IsNullOrEmpty(stripeKey))
22 {
23     throw new ArgumentNullException("Stripe:SecretKey is missing in appsettings.json");
24 }
25
26 StripeConfiguration.ApiKey = stripeKey;
```

Figura 4 - Stripe Program.cs

```
29 // Swagger OpenAPI Documentation
30 builder.Services.AddEndpointsApiExplorer();
31 builder.Services.AddSwaggerGen(c =>
32 {
33     // Swagger OpenAPI Documentation
34     c.SwaggerDoc("v1", new OpenApiInfo
35     {
36         Title = "ISIWebAPI",
37         Version = "v1",
38         Description = "ISI JWT Web API - Ecommerce Store com documentacao OpenAPI JWT",
39         Contact = new OpenApiContact
40         {
41             Name = "Integracao de Sistemas de Informacao 2024/25",
42             Email = "a23502@alunos.ipca.pt",
43             Url = new Uri("https://www.ipca.pt"),
44         },
45     });
```

Figura 5 - Swagger Documentation Program.cs

```
47 // Enabling JWT Bearer authentication
48
49 c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme()
50 {
51     Name = "Authorization",
52     Type = SecuritySchemeType.ApiKey,
53     Scheme = "Bearer",
54     BearerFormat = "JWT",
55     In = ParameterLocation.Header,
56     Description = "JWT Authorization header using the Bearer scheme. \r\n\r\n Enter 'Bearer:'[space] and then your token in the text input below.\r\n\r\nExample: 'Bearer 12345abcdeft'",
57 });
58
59 c.AddSecurityRequirement(new OpenApiSecurityRequirement()
60 {
61     {
62         new OpenApiSecurityScheme
63         {
64             Reference = new OpenApiReference
65             {
66                 Type = ReferenceType.SecurityScheme,
67                 Id = "Bearer"
68             },
69             new string[] {}
70         }
71     }
72 });
```

Figura 6 - JWT Token Program.cs

Services

A pasta *Services*, contém dois ficheiros. Sendo os mesmos, *JwtService.cs* e *UserService.cs*. Ambos os ficheiros servem para auxiliar os *Tokens JWT*.

Como proteções de dados como a adição de *hashes* ou *salts* ou quaisquer outros procedimentos não foram implementados neste projeto, as credenciais dos *users* são verificadas diretamente entre o *username* e *password* inseridas com os dados na base de dados.

JwtService.cs

Este serviço auxilia diretamente os *Tokens JWT*, sendo que é este mesmo serviço que é encarregue de os gerar e atribuir a um *UserID*.

```
23  /// <summary>
24  /// Generates the JWT token
25  /// </summary>
26  /// <param name="user"></param>
27  /// <returns></returns>
    1 reference
28  public string GenerateToken(UserModel user)
29  {
30      var claims = new List<Claim>
31      {
32          new Claim(ClaimTypes.NameIdentifier, user.UserID.ToString()),
33          new Claim(ClaimTypes.Name, user.Username),
34      };
35
36      var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.Key));
37      var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
38
39      var tokenDescriptor = new SecurityTokenDescriptor
40      {
41          Subject = new ClaimsIdentity(claims),
42          Expires = DateTime.UtcNow.AddMinutes(_jwtSettings.TokenExpiryInMinutes),
43          SigningCredentials = credentials,
44          Issuer = _jwtSettings.Issuer,
45          Audience = _jwtSettings.Audience,
46      };
47
48      var tokenHandler = new JwtSecurityTokenHandler();
49      var securityToken = tokenHandler.CreateToken(tokenDescriptor);
50
51      return tokenHandler.WriteToken(securityToken);
52  }
53
```

Figura 7 – JwtServices

UserService

O *UserService* é responsável pela validação dos dados de autenticação introduzidos pelo *user*, quanto aos dados existentes na *database*.

```
20  // <summary>
21  // Validates the user credentials
22  // </summary>
23  // <param name="username"></param>
24  // <param name="password"></param>
25  // <returns></returns>
26  2 references
27  public UserModel ValidateUser(string username, string password)
28  {
29      UserModel user = null;
30
31      using (var connection = new SqlConnection(_configuration.GetConnectionString("ISIWebAPI")))
32      {
33          connection.Open();
34
35          // Query to check if the user exists and password matches
36          var query = @"SELECT UserID, Username, Password FROM Users
37                      WHERE Username = @Username AND Password = @Password";
38
39          using (var command = new SqlCommand(query, connection))
40          {
41              command.Parameters.AddWithValue("@Username", username);
42              command.Parameters.AddWithValue("@Password", password);
43
44              using (var reader = command.ExecuteReader())
45              {
46                  if (reader.Read())
47                  {
48                      user = new UserModel
49                      {
50                          UserID = reader.GetInt32(0),
51                          Username = reader.GetString(1),
52                          // Optionally include the password or other fields if needed
53                      };
54                  }
55              }
56          }
57
58          return user;
59      }
```

Figura 8 – UserService

Models

A camada *Models*, tal como o próprio nome indica, detém os modelos, ou tabelas, da base de dados em SQL. Nesta pasta encontram-se cerca de **20 modelos**, dito isto, apenas 1 modelo será representado. Já que, os modelos **são bastante semelhantes** entre si e repetem o mesmo processo.

```
1  namespace RESTful_API.Models
2  {
3      11 references
4      public class UserModel
5      {
6          4 references
7          public int UserID { get; set; }
8          7 references
9          public string Username { get; set; }
10         5 references
11         public string Email { get; set; }
12         0 references
13         public int Token { get; set; }
14         0 references
15         public DateTime CreatedAt { get; set; }
16         0 references
17         public DateTime UpdatedAt { get; set; }
18         4 references
19         public string Password { get; set; }
20     }
21
22     1 reference
23     public class LoginModel
24     {
25         1 reference
26         public string Username { get; set; }
27         1 reference
28         public string Password { get; set; }
29     }
30 }
```

Figura 9 – UserModel

Para além do *UserModel* completo e igualado às tabelas da base de dados, temos também o *LoginModel* que é utilizado para efeitos de autenticação em conjunto com os serviços de *Tokens JWT*.

Controllers

A camada *Controllers*, é responsável pelas interações do utilizador com a base de dados através da *web*. Tal como nos *Models*, esta camada é bastante repetidora e os ficheiros acabam por ser bastantes semelhantes uns dos outros.

É aqui, que são feitos os “CRUD”, através de *POSTS*, *GETS*, *PUTS* e *DELETES*. No *controller* na Figura 10, podemos ver um *GET Request* do *controller Users*.

```
38  // <summary>
39  // GET Request to fetch all Users
40  // </summary>
41  // <returns></returns>
42  [HttpGet]
43  [Route("getAll[controller]")]
44  0 references
45  public IActionResult GetAllUsers()
46  {
47      try
48      {
49          // Inits the UserModel as a List
50          var userList = new List<UserModel>();
51
52          // Connection string
53          using(var connection = new SqlConnection(_configuration.GetConnectionString(api)))
54          {
55              // Opens the connection and builds the query and executes it
56              connection.Open();
57              var query = "SELECT * FROM Users";
58              using (var command = new SqlCommand(query, connection))
59              {
60                  using (var reader = command.ExecuteReader())
61                  {
62                      while (reader.Read())
63                      {
64                          // Adds fetched fields to the correspondent UserModel fields
65                          userList.Add(new UserModel
66                          {
67                              UserID = reader.GetInt32(0),
68                              Username = reader.GetString(1),
69                              Email = reader.GetString(2),
70                              Password = reader.GetString(3),
71                          });
72                      }
73                  }
74              }
75
76              // If it detects 1+ users on the list it returns Ok 200 status code else its empty
77              if(userList.Count > 0)
78              {
79                  return Ok(userList);
80              }
81              else
82              {
83                  return NotFound(new { Message = "No data found" });
84              }
85          }
86
87          // Error occurred
88          catch (Exception ex)
89          {
90              return StatusCode(500, new { Message = "An error occurred", Error = ex.Message });
91          }
92      }
93  }
```

Figura 10 - UserController (GetAll)

Na seguinte Figura 11, podemos observar um *POST Request* para adicionar um *User*.

```
146  /// <summary>
147  /// POST Request to post a new user
148  /// </summary>
149  /// <returns></returns>
150  [HttpPost]
151  [Route("add[controller]")]
152  0 references
153  public IActionResult AddUser([FromBody] UserModel newUser)
154  {
155      // UserName and email are required and makes sure they are inputted
156      if (newUser == null || string.IsNullOrEmpty(newUser.Username) || string.IsNullOrEmpty(newUser.Email))
157      {
158          return BadRequest(new { Message = "Invalid input. Username and Email are required." });
159      }
160
161      try
162      {
163          using (var connection = new SqlConnection(_configuration.GetConnectionString(api)))
164          {
165              connection.Open();
166
167              // SQL Query for inserting a new user
168              var query = @"INSERT INTO Users (Username, Email, Password)
169                          VALUES (@Username, @Email, @Password);
170                          SELECT SCOPE_IDENTITY();";
171
172              using (var command = new SqlCommand(query, connection))
173              {
174                  // Add parameters to the query
175                  command.Parameters.AddWithValue("@Username", newUser.Username);
176                  command.Parameters.AddWithValue("@Email", newUser.Email);
177                  command.Parameters.AddWithValue("@Password", newUser.Password);
178
179                  // Execute query and get the new user ID
180                  var result = command.ExecuteScalar();
181                  if (result != null)
182                  {
183                      int newUserId = Convert.ToInt32(result);
184                      return CreatedAtAction(nameof(GetUserById), new { userId = newUserId }, new { UserId = newUserId });
185                  }
186                  else
187                  {
188                      return StatusCode(500, new { Message = "User not found. Please try a valid user" });
189                  }
190              }
191          }
192      }
193      catch (Exception ex)
194      {
195          return StatusCode(500, new { Message = "An error occurred while adding the user.", Error = ex.Message });
196      }
197  }
```

Figura 11 - UserController (Post)

De seguida, na Figura 12, temos um *PUT Request* para atualizar um *user* já existente.

```
146 // <summary>
147 // POST Request to post a new user
148 // </summary>
149 // <returns></returns>
150 [HttpPost]
151 [Route("add[controller]")]
152 0 references
153 public IActionResult AddUser([FromBody] UserModel newUser)
154 {
155     // Username and email are required and makes sure they are inputted
156     if (newUser == null || string.IsNullOrEmpty(newUser.Username) || string.IsNullOrEmpty(newUser.Email))
157     {
158         return BadRequest(new { Message = "Invalid input. Username and Email are required." });
159     }
160
161     try
162     {
163         using (var connection = new SqlConnection(_configuration.GetConnectionString(api)))
164         {
165             connection.Open();
166
167             // SQL Query for inserting a new user
168             var query = @"INSERT INTO Users (Username, Email, Password)
169                           VALUES (@Username, @Email, @Password);
170                           SELECT SCOPE_IDENTITY();";
171
172             using (var command = new SqlCommand(query, connection))
173             {
174                 // Add parameters to the query
175                 command.Parameters.AddWithValue("@Username", newUser.Username);
176                 command.Parameters.AddWithValue("@Email", newUser.Email);
177                 command.Parameters.AddWithValue("@Password", newUser.Password);
178
179                 // Execute query and get the new user ID
180                 var result = command.ExecuteScalar();
181                 if (result != null)
182                 {
183                     int newUserId = Convert.ToInt32(result);
184                     return CreatedAtAction(nameof(GetUserById), new { userId = newUserId }, new { UserId = newUserId });
185                 }
186                 else
187                 {
188                     return StatusCode(500, new { Message = "User not found. Please try a valid user" });
189                 }
190             }
191         }
192     }
193     catch (Exception ex)
194     {
195         return StatusCode(500, new { Message = "An error occurred while adding the user.", Error = ex.Message });
196     }
197 }
```

Figura 12 - UserController (Put)

Por fim, na Figura 13, o *DELETE Request* apaga um *user* da base de dados.

```
203 // <summary>
204 // DELETE Request to delete certain user by id from the database
205 // </summary>
206 // <param name="userId"></param>
207 // <returns></returns>
208 [HttpDelete]
209 [Route("delete[controller]/{userId}")]
210 public IActionResult DeleteUser(int userId)
211 {
212     try
213     {
214         using (var connection = new SqlConnection(_configuration.GetConnectionString(api)))
215         {
216             connection.Open();
217             var query = ("DELETE FROM Users WHERE UserID = @UserId");
218
219             using (var command = new SqlCommand(query, connection))
220             {
221                 command.Parameters.AddWithValue("@UserId", userId);
222
223                 var rowsAffected = command.ExecuteNonQuery();
224                 if (rowsAffected > 0)
225                 {
226                     return Ok(new { Message = "User deleted successfully!" });
227                 }
228                 else
229                 {
230                     return StatusCode(500, new { Message = "User not found. Please try a valid user" });
231                 }
232             }
233         }
234     }
235     catch (Exception ex)
236     {
237         return StatusCode(500, new { Message = "An error occurred while deleting the user.", Error = ex.Message });
238     }
239 }
```

Figura 13 - UserController (Delete)

JWT Authentication

A autenticação por *JWT* (*JSON Web Token*), foi implementada dentro do *LoginController* como está evidente na Figura 14.

```
8 [Route("api/[controller]")]
9 [ApiController]
10 1 reference
11 public class LoginController : ControllerBase
12 {
13     private readonly JwtService _jwtService;
14     private readonly UserService _userService;
15
16     0 references
17     public LoginController(JwtService jwtService, UserService userService)
18     {
19         _jwtService = jwtService;
20         _userService = userService;
21     }
22
23     /// <summary>
24     /// POST login request that verifies if the credentials are valid and generates a token for that user
25     /// </summary>
26     /// <param name="login"></param>
27     /// <returns></returns>
28     [HttpPost]
29     [Route("[controller]")]
30     0 references
31     public async Task<IActionResult> Login([FromBody] LoginModel login)
32     {
33         var user = _userService.ValidateUser(login.Username, login.Password);
34
35         if(user == null)
36         {
37             return Unauthorized(new { message = "Invalid credentials" });
38         }
39
40         var token = _jwtService.GenerateToken(user);
41
42         return Ok(new { token });
43     }
44 }
```

Figura 14 - JWT Login Implementation

Para comprovar que a autenticação foi bem implementada, podemos adicionar “[Authorize]” por cima de um *request* para que seja restrito a *users* autorizados. Para efeitos de teste, no *controller* das encomendas, *order*, o request para criar uma encomenda necessita de um user devidamente autorizado. Podemos observá-lo na Figura 15

```
187 187
188 188
189 189
190 190
191 191
192 192
193 193
194 194
195 195
196 196
197 197
198 198
199 199
200 200
201 201
202 202
203 203
204 204
205 205
206 206
207 207
208 208
209 209
210 210
211 211
212 212
213 213
214 214
215 215
216 216
217 217
218 218
219 219
220 220
221 221
222 222
223 223
224 224
225 225
226 226
227 227
228 228
229 229
230 230
231 231
232 232
233 233
234 234
235 235
236 236
237 237
238 238
239 239
240 240
241 241
242 242
243 243
244 244
245 245
246 246
247 247
248 248
249 249
250 250
251 251
252 252
253 253
254 254
255 255
256 256
257 257
258 258
259 259
260 260
261 261
262 262
263 263
264 264
265 265
266 266
267 267
268 268
269 269
270 270
271 271
272 272
273 273
274 274
275 275
276 276
277 277
278 278
279 279
280 280
281 281
282 282
283 283
284 284
285 285
286 286
287 287
288 288
289 289
290 290
291 291
292 292
293 293
294 294
295 295
296 296
297 297
298 298
299 299
300 300
301 301
302 302
303 303
304 304
305 305
306 306
307 307
308 308
309 309
310 310
311 311
312 312
313 313
314 314
315 315
316 316
317 317
318 318
319 319
320 320
321 321
322 322
323 323
324 324
325 325
326 326
327 327
328 328
329 329
330 330
331 331
332 332
333 333
334 334
335 335
336 336
337 337
338 338
339 339
340 340
341 341
342 342
343 343
344 344
345 345
346 346
347 347
348 348
349 349
350 350
351 351
352 352
353 353
354 354
355 355
356 356
357 357
358 358
359 359
360 360
361 361
362 362
363 363
364 364
365 365
366 366
367 367
368 368
369 369
370 370
371 371
372 372
373 373
374 374
375 375
376 376
377 377
378 378
379 379
380 380
381 381
382 382
383 383
384 384
385 385
386 386
387 387
388 388
389 389
390 390
391 391
392 392
393 393
394 394
395 395
396 396
397 397
398 398
399 399
400 400
401 401
402 402
403 403
404 404
405 405
406 406
407 407
408 408
409 409
410 410
411 411
412 412
413 413
414 414
415 415
416 416
417 417
418 418
419 419
420 420
421 421
422 422
423 423
424 424
425 425
426 426
427 427
428 428
429 429
430 430
431 431
432 432
433 433
434 434
435 435
436 436
437 437
438 438
439 439
440 440
441 441
442 442
443 443
444 444
445 445
446 446
447 447
448 448
449 449
450 450
451 451
452 452
453 453
454 454
455 455
456 456
457 457
458 458
459 459
460 460
461 461
462 462
463 463
464 464
465 465
466 466
467 467
468 468
469 469
470 470
471 471
472 472
473 473
474 474
475 475
476 476
477 477
478 478
479 479
480 480
481 481
482 482
483 483
484 484
485 485
486 486
487 487
488 488
489 489
490 490
491 491
492 492
493 493
494 494
495 495
496 496
497 497
498 498
499 499
500 500
501 501
502 502
503 503
504 504
505 505
506 506
507 507
508 508
509 509
510 510
511 511
512 512
513 513
514 514
515 515
516 516
517 517
518 518
519 519
520 520
521 521
522 522
523 523
524 524
525 525
526 526
527 527
528 528
529 529
530 530
531 531
532 532
533 533
534 534
535 535
536 536
537 537
538 538
539 539
540 540
541 541
542 542
543 543
544 544
545 545
546 546
547 547
548 548
549 549
550 550
551 551
552 552
553 553
554 554
555 555
556 556
557 557
558 558
559 559
560 560
561 561
562 562
563 563
564 564
565 565
566 566
567 567
568 568
569 569
570 570
571 571
572 572
573 573
574 574
575 575
576 576
577 577
578 578
579 579
580 580
581 581
582 582
583 583
584 584
585 585
586 586
587 587
588 588
589 589
590 590
591 591
592 592
593 593
594 594
595 595
596 596
597 597
598 598
599 599
600 600
601 601
602 602
603 603
604 604
605 605
606 606
607 607
608 608
609 609
610 610
611 611
612 612
613 613
614 614
615 615
616 616
617 617
618 618
619 619
620 620
621 621
622 622
623 623
624 624
625 625
626 626
627 627
628 628
629 629
630 630
631 631
632 632
633 633
634 634
635 635
636 636
637 637
638 638
639 639
640 640
641 641
642 642
643 643
644 644
645 645
646 646
647 647
648 648
649 649
650 650
651 651
652 652
653 653
654 654
655 655
656 656
657 657
658 658
659 659
660 660
661 661
662 662
663 663
664 664
665 665
666 666
667 667
668 668
669 669
670 670
671 671
672 672
673 673
674 674
675 675
676 676
677 677
678 678
679 679
680 680
681 681
682 682
683 683
684 684
685 685
686 686
687 687
688 688
689 689
690 690
691 691
692 692
693 693
694 694
695 695
696 696
697 697
698 698
699 699
700 700
701 701
702 702
703 703
704 704
705 705
706 706
707 707
708 708
709 709
710 710
711 711
712 712
713 713
714 714
715 715
716 716
717 717
718 718
719 719
720 720
721 721
722 722
723 723
724 724
725 725
726 726
727 727
728 728
729 729
730 730
731 731
732 732
733 733
734 734
735 735
736 736
737 737
738 738
739 739
740 740
741 741
742 742
743 743
744 744
745 745
746 746
747 747
748 748
749 749
750 750
751 751
752 752
753 753
754 754
755 755
756 756
757 757
758 758
759 759
760 760
761 761
762 762
763 763
764 764
765 765
766 766
767 767
768 768
769 769
770 770
771 771
772 772
773 773
774 774
775 775
776 776
777 777
778 778
779 779
780 780
781 781
782 782
783 783
784 784
785 785
786 786
787 787
788 788
789 789
790 790
791 791
792 792
793 793
794 794
795 795
796 796
797 797
798 798
799 799
800 800
801 801
802 802
803 803
804 804
805 805
806 806
807 807
808 808
809 809
810 810
811 811
812 812
813 813
814 814
815 815
816 816
817 817
818 818
819 819
820 820
821 821
822 822
823 823
824 824
825 825
826 826
827 827
828 828
829 829
830 830
831 831
832 832
833 833
834 834
835 835
836 836
837 837
838 838
839 839
840 840
841 841
842 842
843 843
844 844
845 845
846 846
847 847
848 848
849 849
850 850
851 851
852 852
853 853
854 854
855 855
856 856
857 857
858 858
859 859
860 860
861 861
862 862
863 863
864 864
865 865
866 866
867 867
868 868
869 869
870 870
871 871
872 872
873 873
874 874
875 875
876 876
877 877
878 878
879 879
880 880
881 881
882 882
883 883
884 884
885 885
886 886
887 887
888 888
889 889
890 890
891 891
892 892
893 893
894 894
895 895
896 896
897 897
898 898
899 899
900 900
901 901
902 902
903 903
904 904
905 905
906 906
907 907
908 908
909 909
910 910
911 911
912 912
913 913
914 914
915 915
916 916
917 917
918 918
919 919
920 920
921 921
922 922
923 923
924 924
925 925
926 926
927 927
928 928
929 929
930 930
931 931
932 932
933 933
934 934
935 935
936 936
937 937
938 938
939 939
940 940
941 941
942 942
943 943
944 944
945 945
946 946
947 947
948 948
949 949
950 950
951 951
952 952
953 953
954 954
955 955
956 956
957 957
958 958
959 959
960 960
961 961
962 962
963 963
964 964
965 965
966 966
967 967
968 968
969 969
970 970
971 971
972 972
973 973
974 974
975 975
976 976
977 977
978 978
979 979
980 980
981 981
982 982
983 983
984 984
985 985
986 986
987 987
988 988
989 989
990 990
991 991
992 992
993 993
994 994
995 995
996 996
997 997
998 998
999 999
1000 1000
1001 1001
1002 1002
1003 1003
1004 1004
1005 1005
1006 1006
1007 1007
1008 1008
1009 1009
1010 1010
1011 1011
1012 1012
1013 1013
1014 1014
1015 1015
1016 1016
1017 1017
1018 1018
1019 1019
1020 1020
1021 1021
1022 1022
1023 1023
1024 1024
1025 1025
1026 1026
1027 1027
1028 1028
1029 1029
1030 1030
1031 1031
1032 1032
1033 1033
1034 1034
1035 1035
1036 1036
1037 1037
1038 1038
1039 1039
1040 1040
1041 1041
1042 1042
1043 1043
1044 1044
1045 1045
1046 1046
1047 1047
1048 1048
1049 1049
1050 1050
1051 1051
1052 1052
1053 1053
1054 1054
1055 1055
1056 1056
1057 1057
1058 1058
1059 1059
1060 1060
1061 1061
1062 1062
1063 1063
1064 1064
1065 1065
1066 1066
1067 1067
1068 1068
1069 1069
1070 1070
1071 1071
1072 1072
1073 1073
1074 1074
1075 1075
1076 1076
1077 1077
1078 1078
1079 1079
1080 1080
1081 1081
1082 1082
1083 1083
1084 1084
1085 1085
1086 1086
1087 1087
1088 1088
1089 1089
1090 1090
1091 1091
1092 1092
1093 1093
1094 1094
1095 1095
1096 1096
1097 1097
1098 1098
1099 1099
1100 1100
1101 1101
1102 1102
1103 1103
1104 1104
1105 1105
1106 1106
1107 1107
1108 1108
1109 1109
1110 1110
1111 1111
1112 1112
1113 1113
1114 1114
1115 1115
1116 1116
1117 1117
1118 1118
1119 1119
1120 1120
1121 1121
1122 1122
1123 1123
1124 1124
1125 1125
1126 1126
1127 1127
1128 1128
1129 1129
1130 1130
1131 1131
1132 1132
1133 1133
1134 1134
1135 1135
1136 1136
1137 1137
1138 1138
1139 1139
1140 1140
1141 1141
1142 1142
1143 1143
1144 1144
1145 1145
1146 1146
1147 1147
1148 1148
1149 1149
1150 1150
1151 1151
1152 1152
1153 1153
1154 1154
1155 1155
1156 1156
1157 1157
1158 1158
1159 1159
1160 1160
1161 1161
1162 1162
1163 1163
1164 1164
1165 1165
1166 1166
1167 1167
1168 1168
1169 1169
1170 1170
1171 1171
1172 1172
1173 1173
1174 1174
1175 1175
1176 1176
1177 1177
1178 1178
1179 1179
1180 1180
1181 1181
1182 1182
1183 1183
1184 1184
1185 1185
1186 1186
1187 1187
1188 1188
1189 1189
1190 1190
1191 1191
1192 1192
1193 1193
1194 1194
1195 1195
1196 1196
1197 1197
1198 1198
1199 1199
1200 1200
1201 1201
1202 1202
1203 1203
1204 1204
1205 1205
1206 1206
1207 1207
1208 1208
1209 1209
1210 1210
1211 1211
1212 1212
1213 1213
1214 1214
1215 1215
1216 1216
1217 1217
1218 1218
1219 1219
1220 1220
1221 1221
1222 1222
1223 1223
1224 1224
1225 1225
1226 1226
1227 1227
1228 1228
1229 1229
1230 1230
1231 1231
1232 1232
1233 1233
1234 1234
1235 1235
1236 1236
1237 1237
1238 1238
1239 1239
1240 1240
1241 1241
1242 1242
1243 1243
1244 1244
1245 1245
1246 1246
1247 1247
1248 1248
1249 1249
1250 1250
1251 1251
1252 1252
1253 1253
1254 1254
1255 1255
1256 1256
1257 1257
1258 1258
1259 1259
1260 1260
1261 1261
1262 1262
1263 1263
1264 1264
1265 1265
1266 1266
1267 1267
1268 1268
1269 1269
1270 1270
1271 1271
1272 1272
1273 1273
1274 1274
1275 1275
1276 1276
1277 1277
1278 1278
1279 1279
1280 1280
1281 1281
1282 1282
1283 1283
1284 1284
1285 1285
1286 1286
1287 1287
1288 1288
1289 1289
1290 1290
1291 1291
1292 1292
1293 1293
1294 1294
1295 1295
1296 1296
1297 1297
1298 1298
1299 1299
1300 1300
1301 1301
1302 1302
1303 1303
1304 1304
1305 1305
1306 1306
1307 1307
1308 1308
1309 1309
1310 1310
1311 1311
1312 1312
1313 1313
1314 1314
1315 1315
1316 1316
1317 1317
1318 1318
1319 1319
1320 1320
1321 1321
1322 1322
1323 1323
1324 1324
1325 1325
1326 1326
1327 1327
1328 1328
1329 1329
1330 1330
1331 1331
1332 1332
1333 1333
1334 1334
1335 1335
1336 1336
1337 1337
1338 1338
1339 1339
1340 1340
1341 1341
1342 1342
1343 1343
1344 1344
1345 1345
1346 1346
1347 1347
1348 1348
1349 1349
1350 1350
1351 1351
1352 1352
1353 1353
1354 1354
1355 1355
1356 1356
1357 1357
1358 1358
1359 1359
1360 1360
1361 1361
1362 1362
1363 1363
1364 1364
1365 1365
1366 1366
1367 1367
1368 1368
1369 1369
1370 1370
1371 1371
1372 1372
1373 1373
1374 1374
1375 1375
1376 1376
1377 1377
1378 1378
1379 1379
1380 1380
1381 1381
1382 1382
1383 1383
1384 1384
1385 1385
1386 1386
1387 1387
1388 1388
1389 1389
1390 1390
1391 1391
1392 1392
1393 1393
1394 1394
1395 1395
1396 1396
1397 1397
1398 1398
1399 1399
1400 1400
1401 1401
1402 1402
1403 1403
1404 1404
1405 1405
1406 1406
1407 1407
1408 1408
1409 1409
1410 1410
1411 1411
1412 1412
1413 1413
1414 1414
1415 1415
1416 1416
1417 1417
1418 1418
1419 1419
1420 1420
1421 1421
1422 1422
1423 1423
1424 1424
1425 1425
1426 1426
1427 1427
1428 1428
1429 1429
1430 1430
1431 1431
1432 1432
1433 1433
1434 1434
1435 1435
1436 1436
1437 1437
1438 1438
1439 1439
1440 1440
1441 1441
1442 1442
1443 1443
1444 1444
1445 1445
1446 1446
1447 1447
1448 1448
1449 1449
1450 1450
1451 1451
1452 1452
1453 1453
1454 1454
1455 1455
1456 1456
1457 1457
1458 1458
1459 1459
1460 1460
1461 1461
1462 1462
1463 1463
1464 1464
1465 1465
1466 1466
1467 1467
1468 1468
1469 1469
1470 1470
1471 1471
1472 1472
1473 1473
1474 1474
1475 1475
1476 1476
1477 1477
1478 1478
1479 1479
1480 1480
1481 1481
1482 1482
1483 1483
1484 1484
1485 1485
1486 1486
1487 1487
1488 1488
1489 1489
1490 1490
1491 1491
1492 1492
1493 1493
1494 1494
1495 1495
1496 1496
1497 1497
1498 1498
1499 1499
1500 1500
1501 1501
1502 1502
1503 1503
1504 1504
1505 1505
1506 1506
1507 1507
1508 1508
1509 1509
1510 1510
1511 1511
1512 151
```


Com recurso à aplicação ***Insomnia***, que será mencionada mais detalhadamente brevemente, podemos confirmar a correta e o funcionamento desta funcionalidade. Na Figura 16, utilizamos uma combinação de *username* e *password* válidas para efeitos de login, e a resposta, se válida, será de um *token*.

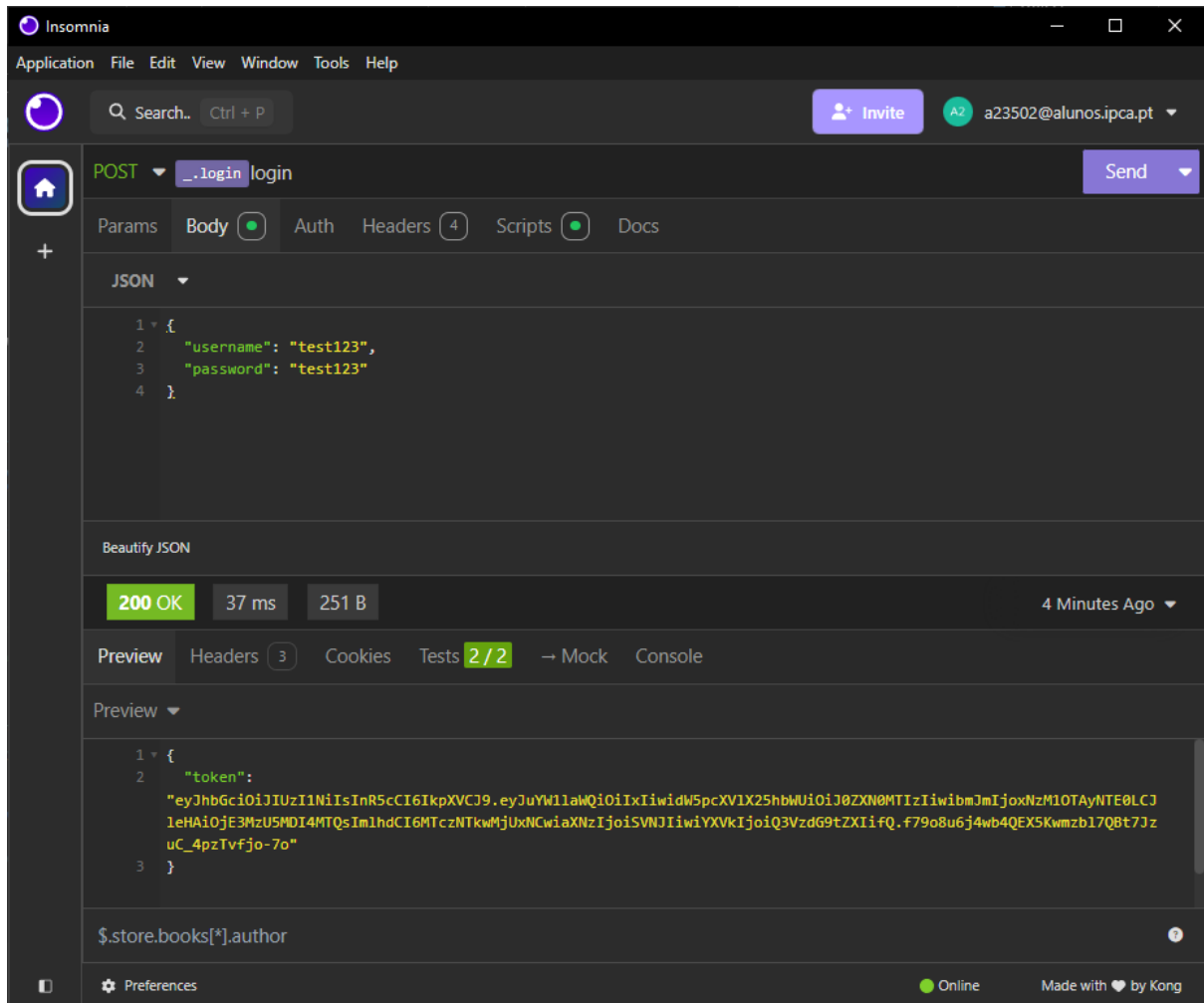


Figura 16 - Login Test

Tal como esperado, o resultado **200 OK** reflete um *request* bem-sucedido, tal como a resposta do *token* que foi atribuído ao *user*.

Por fim, na Figura 17, utilizamos o *user* autorizado para criar uma *order*. Para que seja possível, adicionamos o *token* deste *user* ao campo **Auth** na aplicação *Insomnia*.

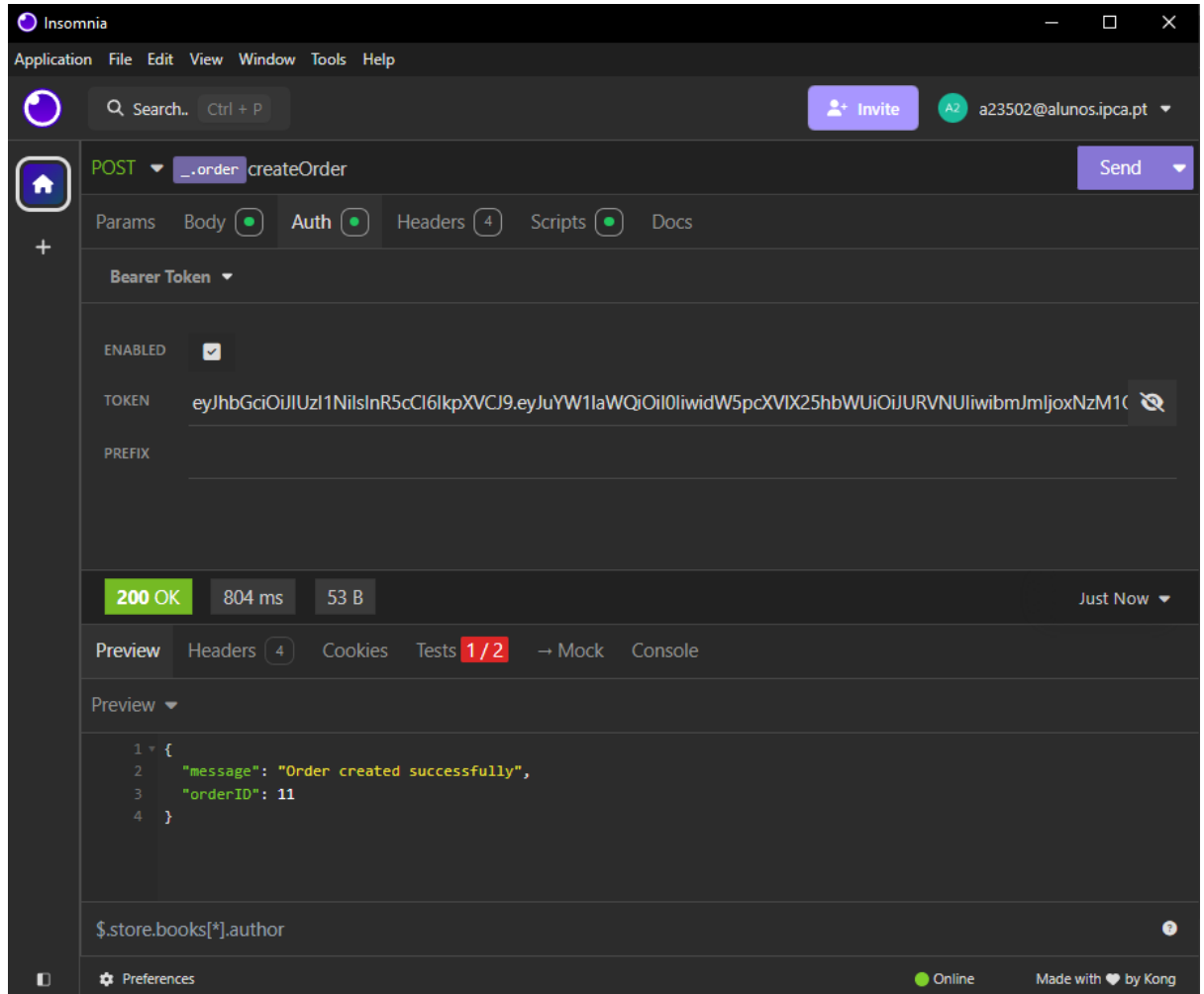


Figura 17 - Authorized User teste

Integração Externa – Stripe

Como integração externa, foi decidido recorrer ao serviço de pagamentos **Stripe**. Uma vez que esta aplicação modela uma loja online, faria mais que sentido integrar um serviço de pagamentos externo. Dada a facilidade de interações com a sua *API*, recorremos então a este serviço.

Na Figura 18 e 19, está presente o *controller* encarregue dos Pagamentos via *Stripe*.

```
27 // <summary>
28 // POST Request to create a payment intent with Stripe
29 // </summary>
30 // <param name="orderId"></param>
31 // <returns></returns>
32 [HttpPost]
33 [Route("createPaymentIntentComplete")]
34 public async Task<ActionResult> CreatePaymentIntentComplete([FromBody] CreatePaymentIntentRequest request)
35 {
36     var order = GetOrderFromDb(request.OrderID);
37     if (order == null)
38     {
39         return NotFound("Order not found");
40     }
41
42     var paymentMethod = GetPaymentMethodFromDb(request.PaymentMethodID);
43     if (paymentMethod == null)
44     {
45         return BadRequest("Invalid PaymentMethodID");
46     }
47
48     if(string.IsNullOrEmpty(paymentMethod.MethodName))
49     {
50         return BadRequest("Payment method name is required");
51     }
52
53     try
54     {
55         // Create the payment intent
56         var options = new PaymentIntentCreateOptions
57         {
58             Amount = (long)(order.Total * 100),
59             Currency = "eur",
60             PaymentMethodTypes = new List<string> { paymentMethod.MethodName },
61             Metadata = new Dictionary<string, string>
62             {
63                 {"OrderID", order.OrderID.ToString() },
64                 {"UserID", order.UserID.ToString() }
65             },
66         };
67     }
```

Figura 18 - Controller Stripe

```
68     var service = new PaymentIntentService();
69     var paymentIntent = await service.CreateAsync(options);
70
71     // Save the PaymentIntent ID in the payments table
72     SavePaymentIntentToDb(order.OrderID, paymentIntent.Id, order.Total, paymentMethod.PaymentMethodID);
73
74     //return Ok(new { ClientSecretCredential = paymentIntent.ClientSecret, });
75     return Ok(new { ClientSecretCredential = paymentIntent.ClientSecret, message = "Success", paymentMethod.MethodName });
76 }
77
78 catch (Exception ex)
79 {
80     return BadRequest(ex.Message);
81 }
82 }
```

Figura 19 - Controller Stripe (1)

Este controller, com recurso a outras funções que fazem interações diretas com a base de dados tais como recolher dados de encomendas, guardar o ID do pagamento do *Stripe* na nossa base de dados e atribuí-lo à encomenda correspondente, atualizar *status* dos pagamentos e recolher os métodos de pagamentos disponíveis na base de dados, é responsável pela verificação de dados e de iniciar um pagamento de acordo com os parâmetros do *Stripe*.

SOAP API

Quanto à SOAP API, foi decidido fazer apenas o *controller* dos pagamentos (“interno”, não é o do *Stripe*). Na Figura 20 podemos ver a estrutura do projeto **SOAP-API**

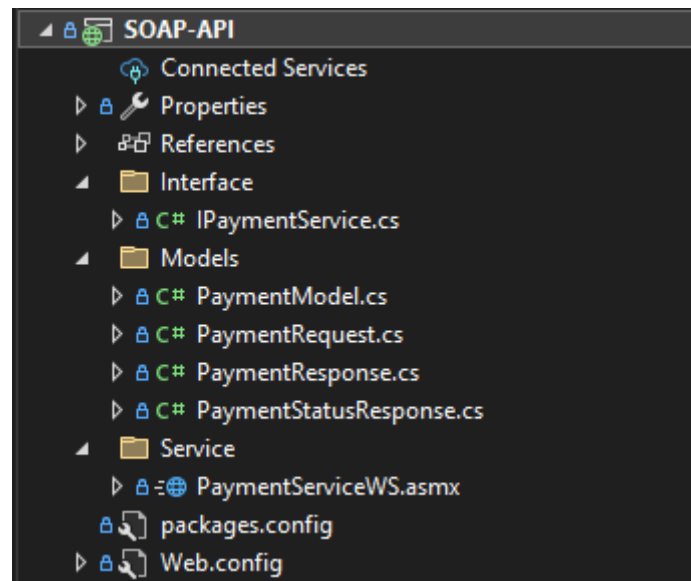


Figura 20 - SOAP API Estrutura

Dentro do projeto estão presentes os *Models* necessários e o *PaymentServiceWS.asmx* que trata devidamente dos *WebMethods*.

Models

Os *Models* presentes na *SOAP API*, são, tal como na *RESTful*, modelos de dados da base de dados que representam as tabelas da mesma. Ou, neste caso, “preparam” modelos necessários para interagir com os *requests*. Nas Figuras 21, 22, 23 e 24, estarão disponíveis os *Models* utilizados.

```
6  namespace SOAP_API.Models
7  {
8      0 references
9      public class PaymentModel
10     {
11         0 references
12         public int PaymentID { get; set; }
13         0 references
14         public int OrderID { get; set; }
15         0 references
16         public int PaymentStatusID { get; set; }
17         0 references
18         public int PaymentMethodID { get; set; }
19         0 references
20         public decimal Amount { get; set; }
21         0 references
22         public DateTime PaymentDate { get; set; }
23     }
24 }
```

Figura 21 - SOAP PaymentModel

```
6  namespace SOAP_API.Models
7  {
8      1 reference
9      public class PaymentResponse
10     {
11         0 references
12         public bool Success { get; set; }
13         0 references
14         public string PaymentIntentID { get; set; }
15         0 references
16         public string Message { get; set; }
17     }
18 }
```

Figura 22 - SOAP PaymentResponseModel

```
7  namespace SOAP_API.Models
8  {
9      1 reference
10     public class PaymentRequest
11     {
12         0 references
13         public int OrderID { get; set; }
14         0 references
15         public int PaymentMethodID { get; set; }
16         0 references
17         public decimal Amount { get; set; }
18     }
19 }
```

Figura 23 - SOAP PaymentResponseModel

```
6  namespace SOAP_API.Models
7  {
8      1 reference
9      public class PaymentStatusResponse
10     {
11         0 references
12         public string Status { get; set; }
13         0 references
14         public string Message { get; set; }
15     }
16 }
```

Figura 24 - SOAP PaymentStatusResponse

Service

Na pasta *Service*, está presente o ficheiro *PaymentServiceWS.asmx*. Serviço este que é responsável por todos os *WebMethods* e as suas interações com a base de dados. Nas próximas Figuras 25 e 26, podemos observar todos os *WebMethods*.

```

9 namespace SOAP_API.Service
10 {
11     [WebService(Namespace = "http://isiwebapi.com/payments")]
12     [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
13     public class PaymentServiceWS : WebService
14     {
15         // Connection string to the db
16         private readonly string _connectionString = "Data Source=192.168.50.50,1433;Dat
17
18         /// <summary>
19         /// Method to make a payment
20         /// </summary>
21         /// <param name="orderId"></param>
22         /// <param name="amount"></param>
23         /// <param name="paymentMethodId"></param>
24         /// <returns></returns>
25         [WebMethod]
26         public string MakePayment(int orderId, decimal amount, int paymentMethodId) ...
27
28         /// <summary>
29         /// Method to get all available payment methods
30         /// </summary>
31         /// <returns></returns>
32         [WebMethod]
33         public List<string> GetAllPaymentMethods() ...
34
35         /// <summary>
36         /// Method to validate if a payment method exists
37         /// </summary>
38         /// <param name="paymentMethodId"></param>
39         /// <returns></returns>
40         [WebMethod]
41         public string ValidatePaymentMethod(int paymentMethodId) ...
42     }

```

Figura 25 - PaymentServiceWS

```

123     /// <summary>
124     /// Method to refund a payment
125     /// </summary>
126     /// <param name="paymentId"></param>
127     /// <returns></returns>
128     [WebMethod]
129     public string RefundPayment(int paymentId) ...
130
131     /// <summary>
132     /// Method to get payment details
133     /// </summary>
134     /// <param name="paymentId"></param>
135     /// <returns></returns>
136     [WebMethod]
137     public string GetPaymentDetails(int paymentId) ...
138
139     /// <summary>
140     /// Method to get payment status
141     /// </summary>
142     /// <param name="paymentId"></param>
143     /// <returns></returns>
144     [WebMethod]
145     public string GetPaymentStatus(int paymentId) ...
146 }

```

Figura 26 - PaymentServiceWS(1)

Novamente, dado que todos os estes métodos recorrem ao mesmo processo repetidamente com algumas modificações, será apenas exibido com detalhe apenas dois *WebMethods*. Sendo o primeiro, o método para criar um pagamento, como será exibido na Figura 27.

```
18  <summary>
19  </summary>
20  </summary>
21  <param name="orderId"></param>
22  <param name="amount"></param>
23  <param name="paymentMethodId"></param>
24  </returns>
25  [WebMethod]
26  public string MakePayment(int orderId, decimal amount, int paymentMethodId)
27  {
28      using (SqlConnection connection = new SqlConnection(_connectionString))
29      {
30          try
31          {
32              connection.Open();
33
34              // Check if order exists
35              SqlCommand checkOrderCmd = new SqlCommand("SELECT COUNT(*) FROM Orders WHERE OrderID = @OrderId", connection);
36              checkOrderCmd.Parameters.AddWithValue("@OrderId", orderId);
37              int orderExists = (int)checkOrderCmd.ExecuteScalar();
38
39              if (orderExists == 0)
40              {
41                  return "Order not found.";
42              }
43
44              // Insert payment
45              SqlCommand insertPaymentCmd = new SqlCommand(
46                  @"INSERT INTO Payments (OrderID, PaymentStatusID, PaymentMethodID, Amount, PaymentDate)
47                  VALUES (@OrderId, 1, @PaymentMethodId, @Amount, GETDATE());
48                  SELECT SCOPE_IDENTITY();",
49                  connection
50              );
51
52              insertPaymentCmd.Parameters.AddWithValue("@OrderId", orderId);
53              insertPaymentCmd.Parameters.AddWithValue("@PaymentMethodId", paymentMethodId);
54              insertPaymentCmd.Parameters.AddWithValue("@Amount", amount);
55
56              int paymentId = Convert.ToInt32(insertPaymentCmd.ExecuteScalar());
57              return $"Payment successful. Payment ID: {paymentId}";
58          }
59          catch (Exception ex)
60          {
61              return $"Error: {ex.Message}";
62          }
63      }
64  }
```

Figura 27 - SOAP MakePayment

Este método aceita como parâmetros o *orderId*, *amount* e o *paymentMethodId* para efetuar as necessárias operações com a base de dados.

O próximo *WebMethod*, será o *RefundPayment*. Onde este efetua um novo pagamento com o valor equivalente ao da *order* só que negativo. Com o intuito de simular um reembolso. O mesmo está explícito na seguinte Figura 28.

```

123  <summary>
124  </summary>
125  </summary>
126  <param name="paymentId"></param>
127  </returns></returns>
128  [WebMethod]
129  0 references
130  public string RefundPayment(int paymentId)
131  {
132      using (SqlConnection connection = new SqlConnection(_connectionString))
133      {
134          try
135          {
136              connection.Open();
137
138              // Get the original payment details
139              SqlCommand getPaymentCmd = new SqlCommand(
140                  @"SELECT OrderID, PaymentMethodID, Amount
141                  FROM Payments WHERE PaymentID = @PaymentId", connection);
142              getPaymentCmd.Parameters.AddWithValue("@PaymentId", paymentId);
143
144              SqlDataReader reader = getPaymentCmd.ExecuteReader();
145              if (!reader.Read())
146              {
147                  return "Payment not found.";
148              }
149
150              int orderId = Convert.ToInt32(reader["OrderID"]);
151              int paymentMethodId = Convert.ToInt32(reader["PaymentMethodID"]);
152              decimal amount = Convert.ToDecimal(reader["Amount"]);
153
154              reader.Close();
155
156              // Insert a negative payment record
157              SqlCommand refundCmd = new SqlCommand(
158                  @"INSERT INTO Payments (OrderID, PaymentStatusID, PaymentMethodID, Amount, PaymentDate)
159                  VALUES (@OrderId, 2, @PaymentMethodId, @RefundAmount, GETDATE());",
160                  connection);
161
162              refundCmd.Parameters.AddWithValue("@OrderId", orderId);
163              refundCmd.Parameters.AddWithValue("@PaymentMethodId", paymentMethodId);
164              refundCmd.Parameters.AddWithValue("@RefundAmount", -amount);
165
166              int rowsAffected = refundCmd.ExecuteNonQuery();
167              return rowsAffected > 0 ? "Refund processed successfully." : "Refund failed.";
168          }
169          catch (Exception ex)
170          {
171              return $"Error: {ex.Message}";
172          }
173      }
174  }

```

Figura 28 - SOAP RefundPayment

Testes

Como já foi referido, os testes de aplicação foram realizados com recurso à ferramenta **Insomnia**. Esta ferramenta facilita a gestão dos *requests*. Possibilita nos de os organizar, adicionar *bodies*, *auth tokens*, adicionar *variáveis*, recorrer a todos os *requests* existentes e entre outros.

Insomnia

Para facilitar ainda mais o processo, recorreremos à utilização de **variáveis globais**. Estas variáveis encurtam um *url* de acesso para uma variável com um nome à escolha. Todas as variáveis utilizadas encontram-se disponíveis na próxima Figura 29.

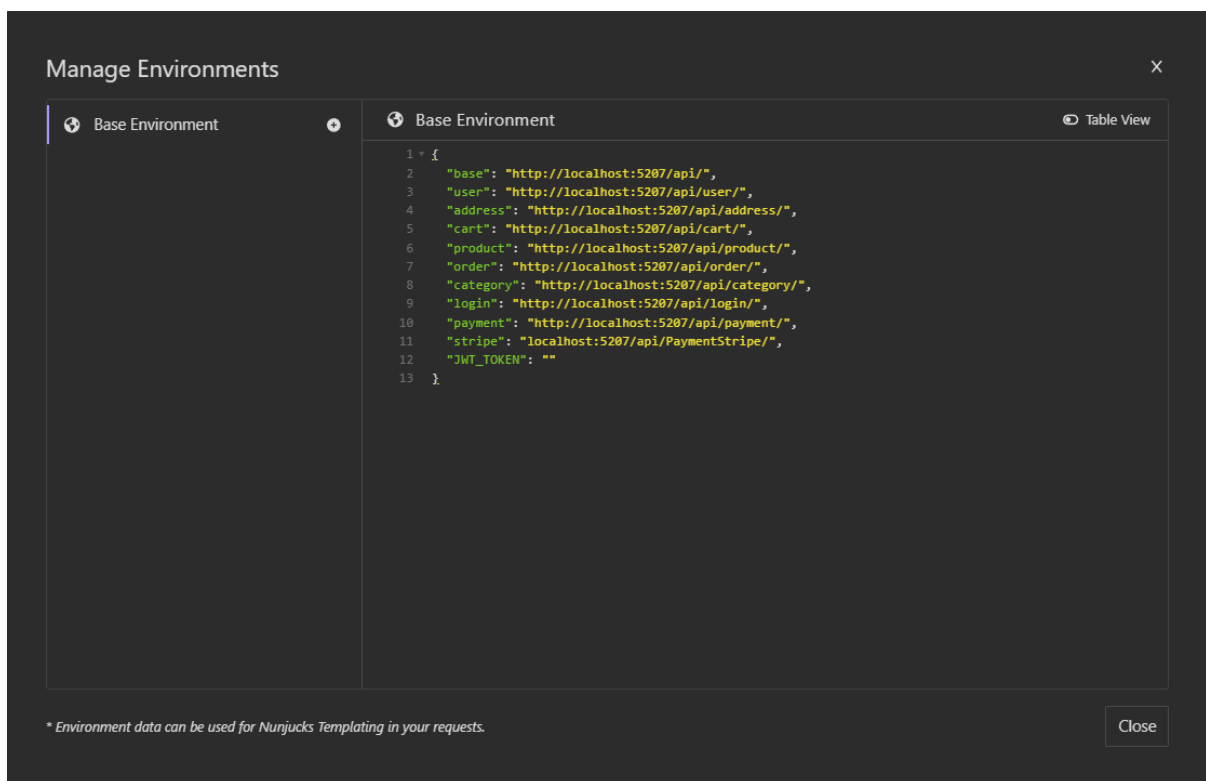


Figura 29 - Insomnia Variáveis

Quanto à estrutura, todos os *requests*, sejam eles *RESTful* ou *SOAP*, foram todos estruturados e agrupados por modelo como está evidente na Figura 30.

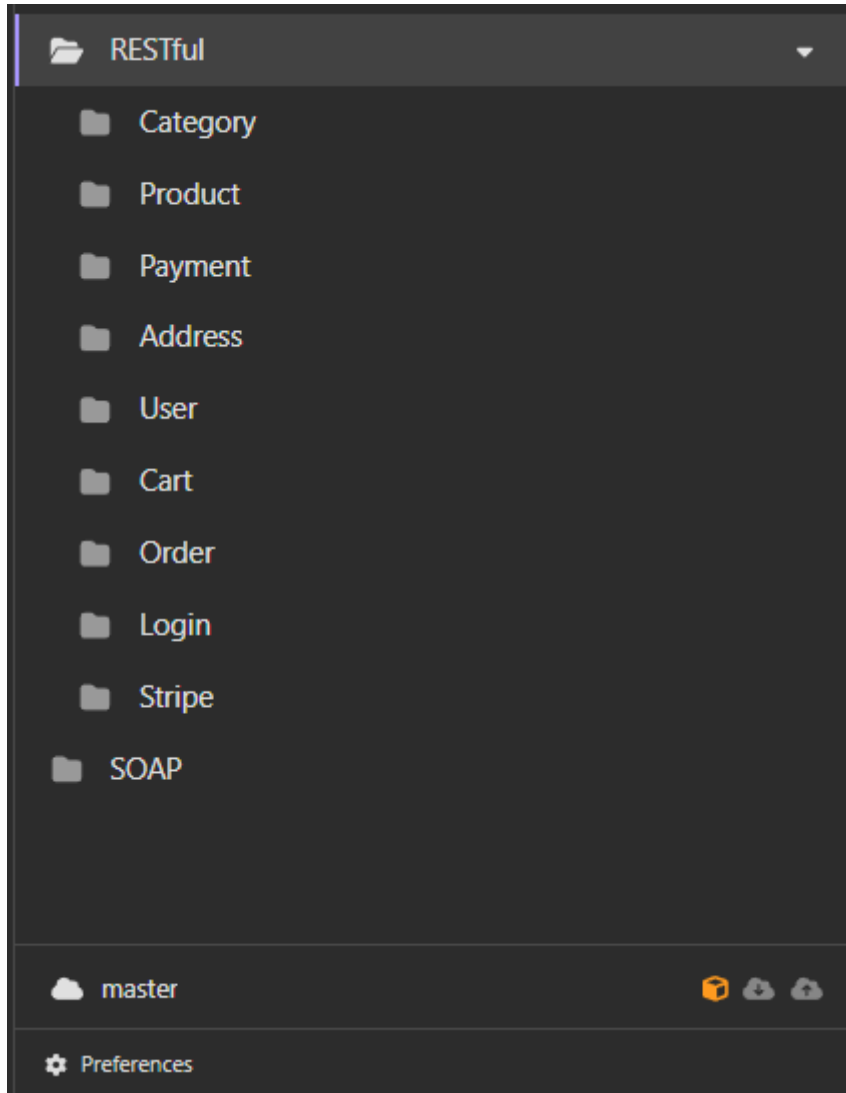
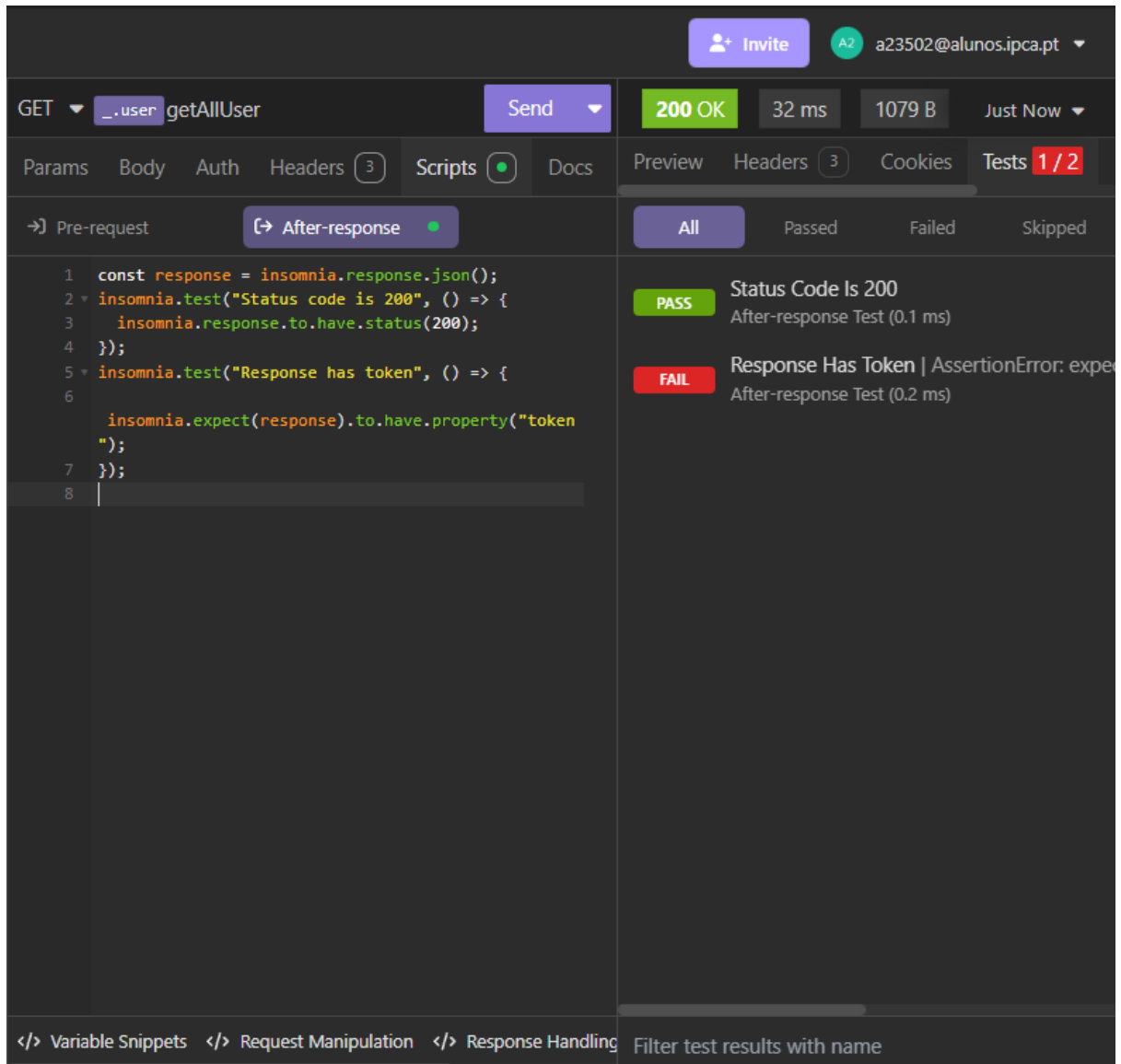


Figura 30 - Insomnia Estrutura

Por fim, foram feitos verdadeiros **Testes** com recurso à aceitação de *scripts* que estão ao dispor do utilizador. Como podemos presenciar na seguinte Figura 31, foi efetuado um teste com *script* que atua **depois** da resposta do *request*. Onde este **passa** o teste se o resultado do *request* for de **Status Code: 200 OK**.



GET **_user** getAllUser **Send** **200 OK** 32 ms 1079 B Just Now

Params Body Auth Headers (3) **Scripts** (0) Docs Preview Headers (3) Cookies **Tests 1/2**

→ Pre-request [→ After-response ●]

```

1 const response = insomnia.response.json();
2 insomnia.test("Status code is 200", () => {
3   insomnia.response.to.have.status(200);
4 });
5 insomnia.test("Response has token", () => {
6   insomnia.expect(response).to.have.property("token");
7 });
8

```

All Passed Failed Skipped

PASS Status Code Is 200
After-response Test (0.1 ms)

FAIL Response Has Token | AssertionError: expected response to have property 'token'
After-response Test (0.2 ms)

</> Variable Snippets </> Request Manipulation </> Response Handling Filter test results with name

Figura 31 - Insomnia Test

Cloud

Apesar de não estar devidamente expresso neste relatório nem no projeto, houve uma inicial base de dados na *cloud* da *Azure*.

Contudo, e infelizmente, os créditos de **Estudante** foram todos utilizados acidentalmente até ao esgotamento, como podemos confirmar na seguinte Figura 32.

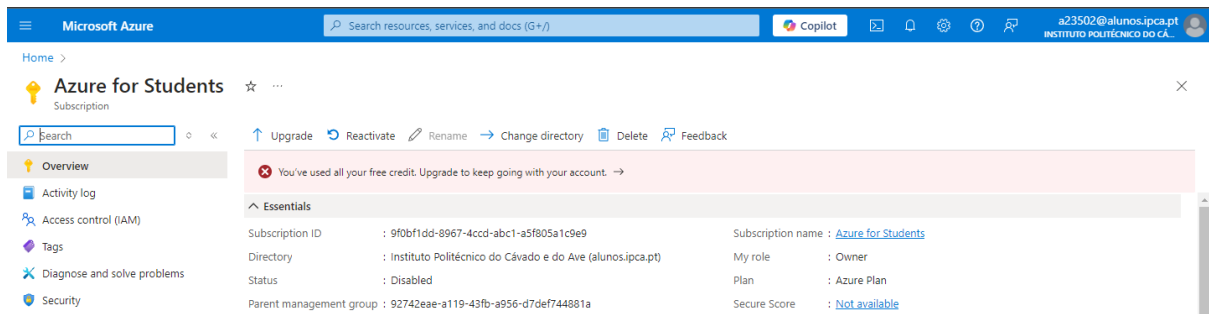


Figura 32 - Azure Painel

Mesmo não estando ativa nem presente no projeto, fica aqui, na figura 33, o *SQL Server* no painel da *Microsoft Azure*.

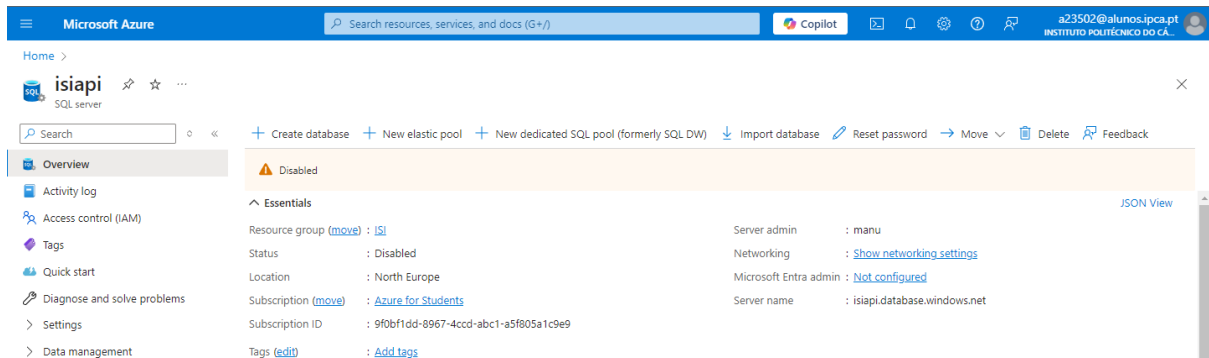


Figura 33 - Azure SQL Database

Conclusão

Em suma, este projeto resultou na criação de uma *API* funcional e segura para um sistema de *e-commerce*, integrando tecnologias modernas como autenticação *JWT* e serviços externos como *Stripe*. A *RESTful API* e a *SOAP API*, atendem a diferentes necessidades de integração.

A documentação detalhada e os testes realizados reforçam a confiabilidade do sistema, que cumpre os objetivos propostos e está preparado para futuras expansões. Este trabalho consolidou boas práticas de desenvolvimento e será uma base sólida para projetos futuros.

Referências

<https://www.youtube.com/watch?v=V5ooPC3wBWl>

<https://www.youtube.com/watch?v=3Op008xgFjw>

<https://www.youtube.com/watch?v=ZVkJrw37j1Q>

<https://docs.stripe.com/payments/payment-intents>

<https://stackoverflow.com/>

<https://chatgpt.com/>

<https://portal.azure.com/>