

GetawayGo

Non-functional requirements implementation

Fontys University of Applied Sciences

Anna Kadurina
16/01/2025

Table of Contents

Introduction.....	1
Performance	2
Efficient Hosting Resources.....	2
Load and Stress Testing	2
Scalability	3
App Services	3
Microservices Architecture	3
Service Bus Integration	4
API Gateway	4
Distributed Resources	5
Security	6
Authentication and Authorization	6
Data Protection.....	7
Vulnerability Testing.....	7
Monitoring and Alerts	7
Usability.....	8
User Interface Design.....	8
Maintainability.....	9
Microservices Architecture	9
Logging and Monitoring	9
CI/CD Pipeline Testing	9
Reliability.....	11
Disaster Recovery	11
Conclusion	11

Introduction

This document outlines the implementation of non-functional requirements (NFRs) in the GetawayGo project. These requirements ensure the system meets quality attributes such as performance, scalability, security, usability, maintainability, and reliability. The measures described below demonstrate how these attributes have been incorporated into the design, development, and deployment of GetawayGo to provide a robust and efficient platform for users.

Performance

Efficient Hosting Resources

The use of Azure Static Web Apps for frontend and Azure App Services for backend microservices provides optimized, high-performance hosting without requiring additional infrastructure such as load balancers. These services distribute workloads efficiently and ensure fast response times.

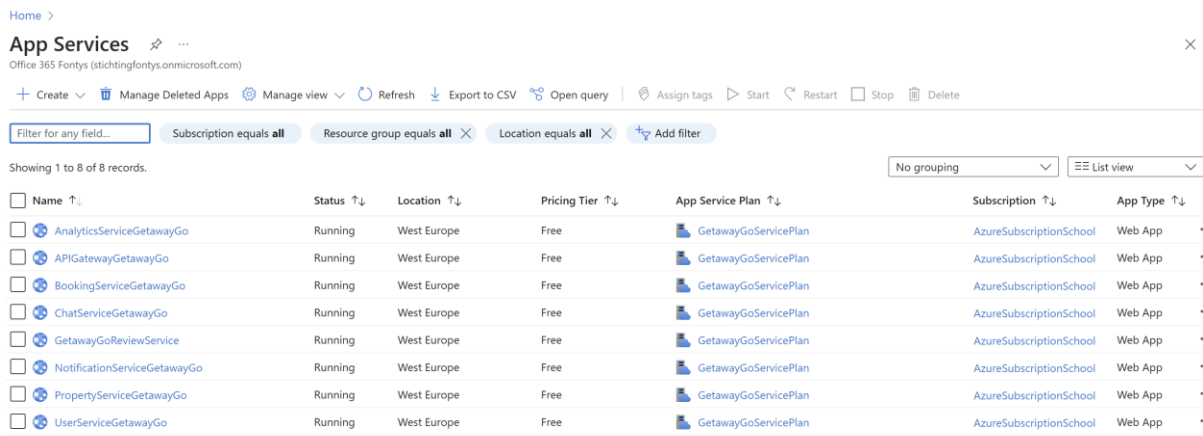


Figure 1 – Overview of App Services

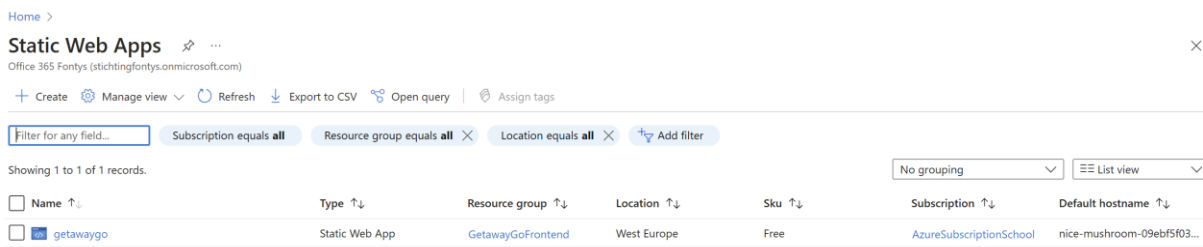


Figure 2 – Overview of Static Web Apps

Load and Stress Testing

Azure Load Testing is regularly employed to evaluate system performance under varying loads identifying bottlenecks and ensuring the system meets expected response times. Stress testing is conducted to assess the system’s behaviour under extreme conditions.

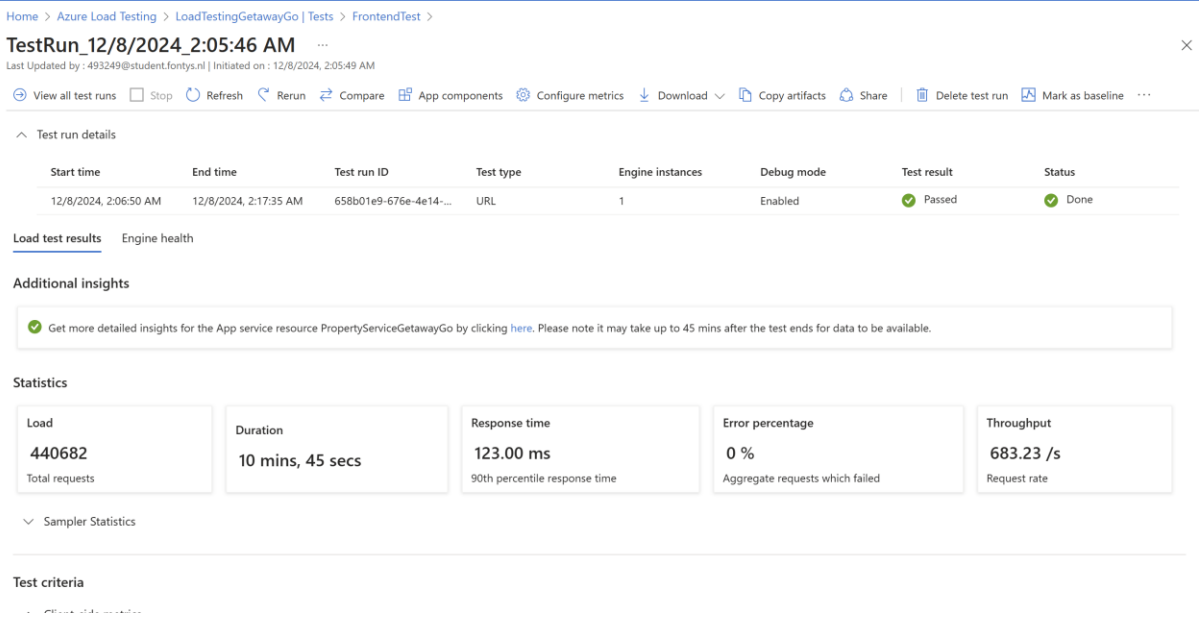


Figure 3 – Load testing

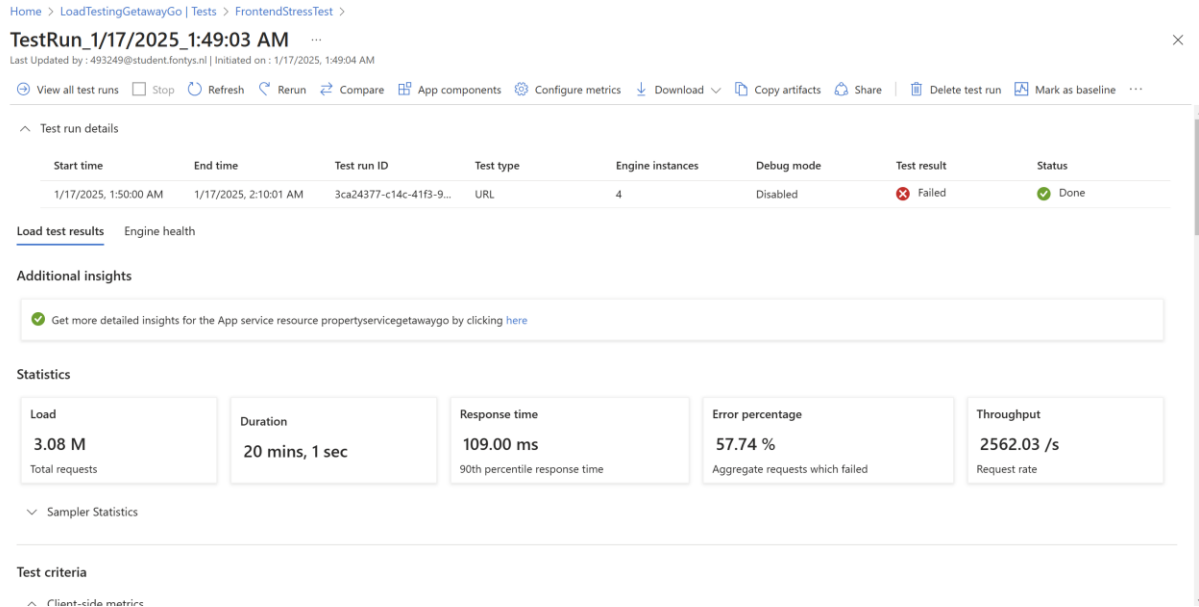


Figure 4 – Stress testing

Scalability

App Services

App Services automatically scale based on demand, thus eliminating manual intervention.

Microservices Architecture

The system is divided into independent microservices each responsible for specific functionalities (see all microservices on Figure 1). This architecture allows individual services to scale independently based on workload.

Service Bus Integration

Azure Service Bus facilitates communication between microservices, enabling decoupled and asynchronous processing. This design prevents bottlenecks and ensures scalability.

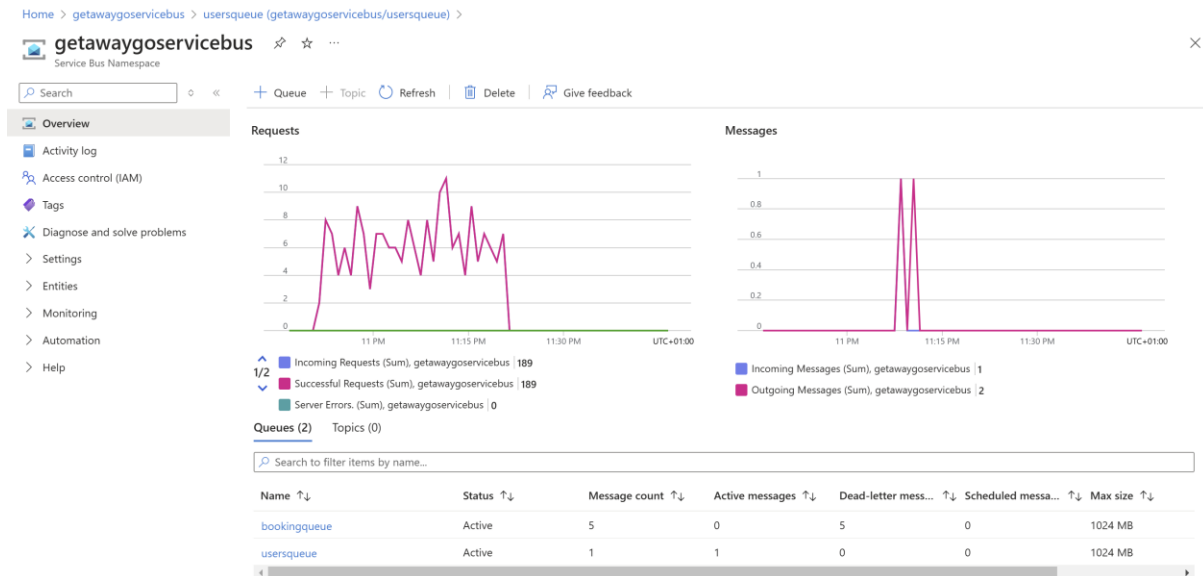


Figure 5 – Service bus

API Gateway

The API Gateway acts as centralized entry point for all client requests, routing traffic to appropriate microservices. Centralized traffic management ensures seamless scaling without overloading individual microservices.

```
1  {
2    "Routes": [
3      {
4        "DownstreamPathTemplate": "/api/user{everything}",
5        "DownstreamScheme": "https",
6        "DownstreamHostAndPorts": [
7          {
8            "Host": "userservicegetawaygo.azurewebsites.net",
9            "Port": 443
10         }
11       ],
12       "UpstreamPathTemplate": "/user/{everything}",
13       "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ]
14     },
15     {
16       "DownstreamPathTemplate": "/api/property/{everything}",
17       "DownstreamScheme": "https",
18       "DownstreamHostAndPorts": [
19         {
20           "Host": "propertyservicegetawaygo.azurewebsites.net",
21           "Port": 443
22         }
23       ],
24       "UpstreamPathTemplate": "/property/{everything}",
25       "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ]
26     },
27     {
28       "DownstreamPathTemplate": "/api/booking/{everything}",
29       "DownstreamScheme": "https",
30       "DownstreamHostAndPorts": [
31         {
32           "Host": "bookingservicegetawaygo.azurewebsites.net",
33           "Port": 443
34         }
35       ],
36       "UpstreamPathTemplate": "/booking/{everything}",
37       "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ]
38     },
39     {
40       "DownstreamPathTemplate": "/api/analytics/{everything}",
41       "DownstreamScheme": "https",
42       "DownstreamHostAndPorts": [
43         {
44           "Host": "analyticsservicegetawaygo.azurewebsites.net",
45           "Port": 443
46         }
47       ],
48     }
49   ]
50 }
```

Figure 6 – API Gateway Configuration

Distributed Resources

Separate servers and databases for each microservice prevent resource contention, ensuring the system scales efficiently with user growth.

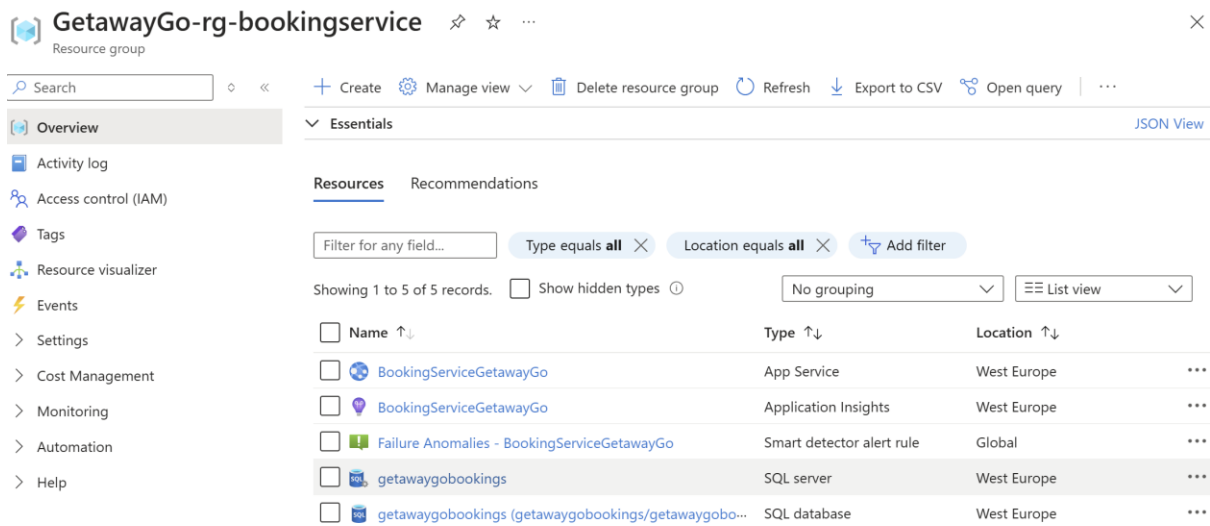


Figure 7 – Example of a service with its own SQL Server

Security

Authentication and Authorization

OAuth 2.0 ensures secure user authentication and access control. Role-based access control (RBAC) is implemented to restrict access to sensitive operations based on user roles (guests, hosts, admins).

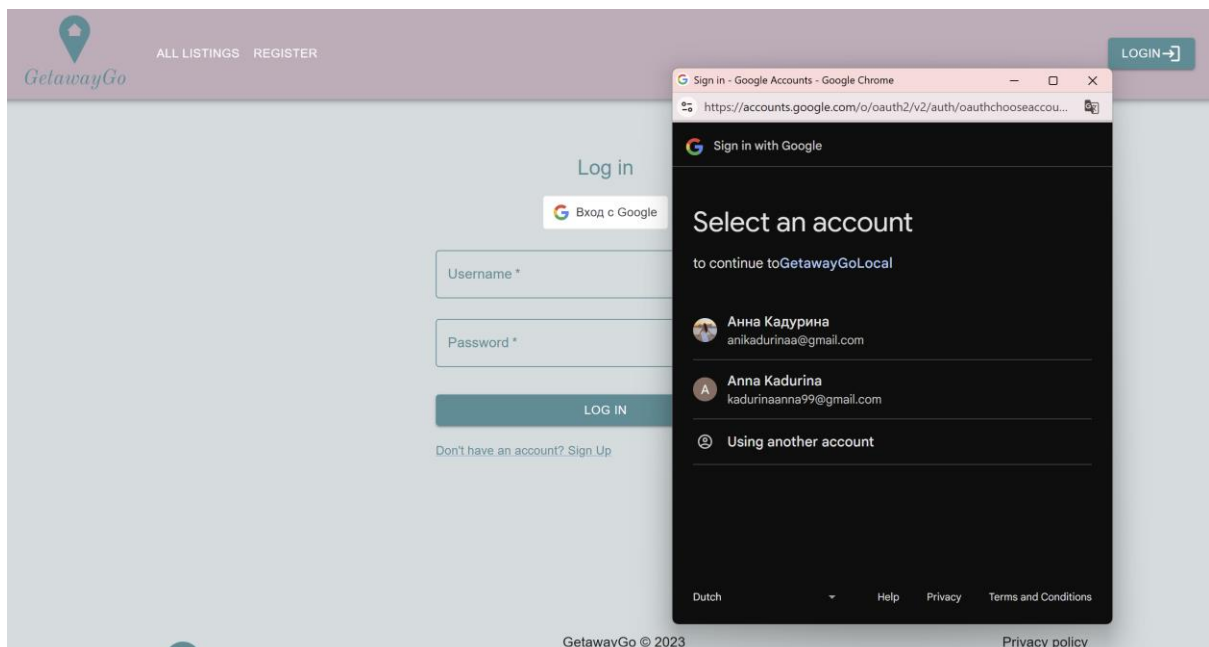


Figure 8 – OAuth 2.0

Data Protection

User passwords are securely hashed and all data is on secure cloud servers. GDPR compliance ensures that personal data is collected, stored, and processed responsibly.

Results		Messages						
	UserID	FirstName	LastName	Email	PasswordHash	PhoneNumber	DateOfBirth	CreatedAt
1	1	Anche	Kadurina	anikadurinaa@gmail.com	wVPNIYplYRfhpbwbcNH5fFuioWwBXXCrU670ZepS45w=	123-456-7890	2002-09-09	2024-10-13 20:21:24.613

Figure 9 – Password Hashing

Vulnerability Testing

Regular vulnerability scans are performed using Snyk and OWASP ZAP in CI/CD pipelines identify and address potential security issues.

Monitoring and Alerts

Application Insights and Sentry.io provide real-time monitoring and alerting for suspicious activity or anomalies. Emails are sent to me in case of a problem.

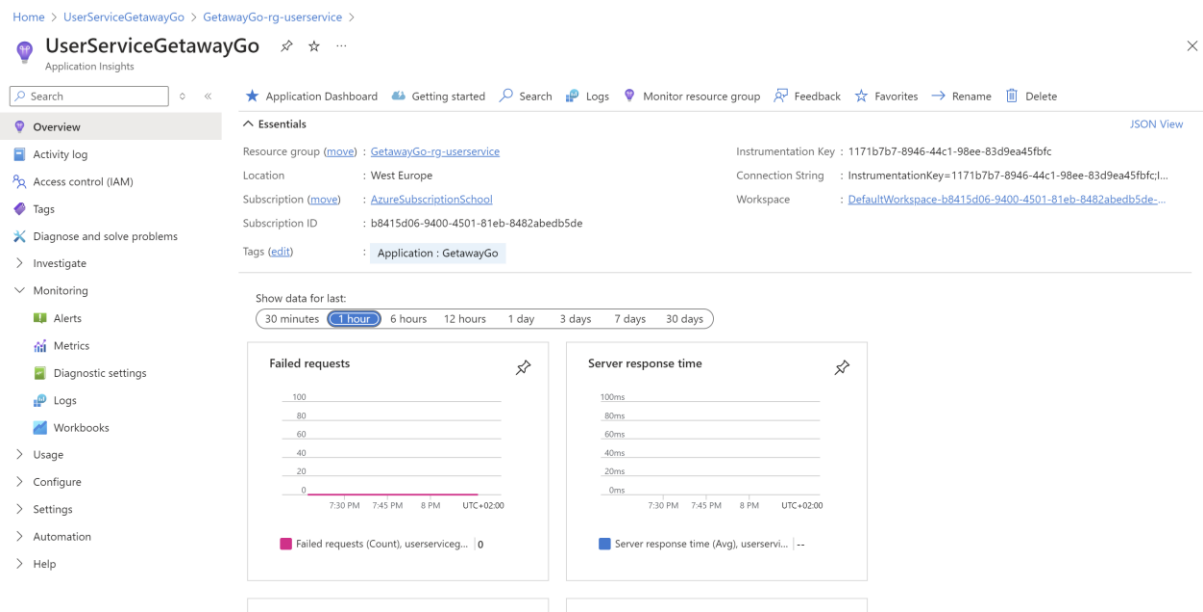


Figure 10 – Application Insights

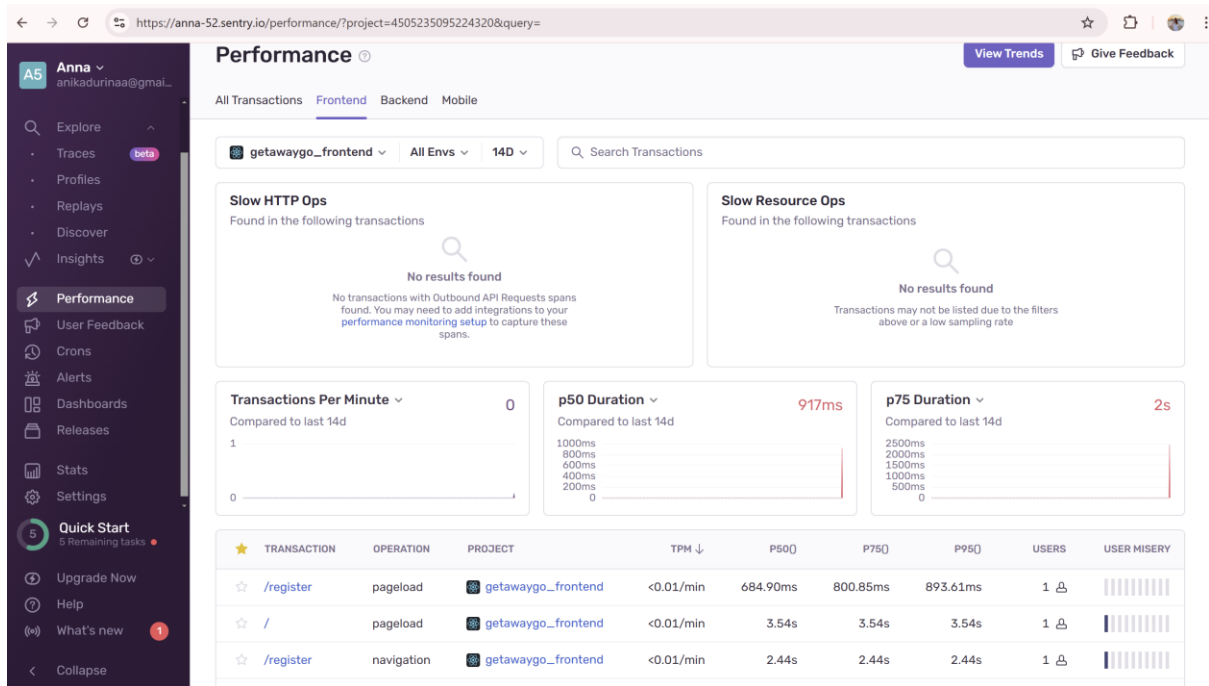


Figure 11 – Sentry.io monitoring dashboard

Usability

User Interface Design

The interface is designed with focus on simplicity and accessibility.

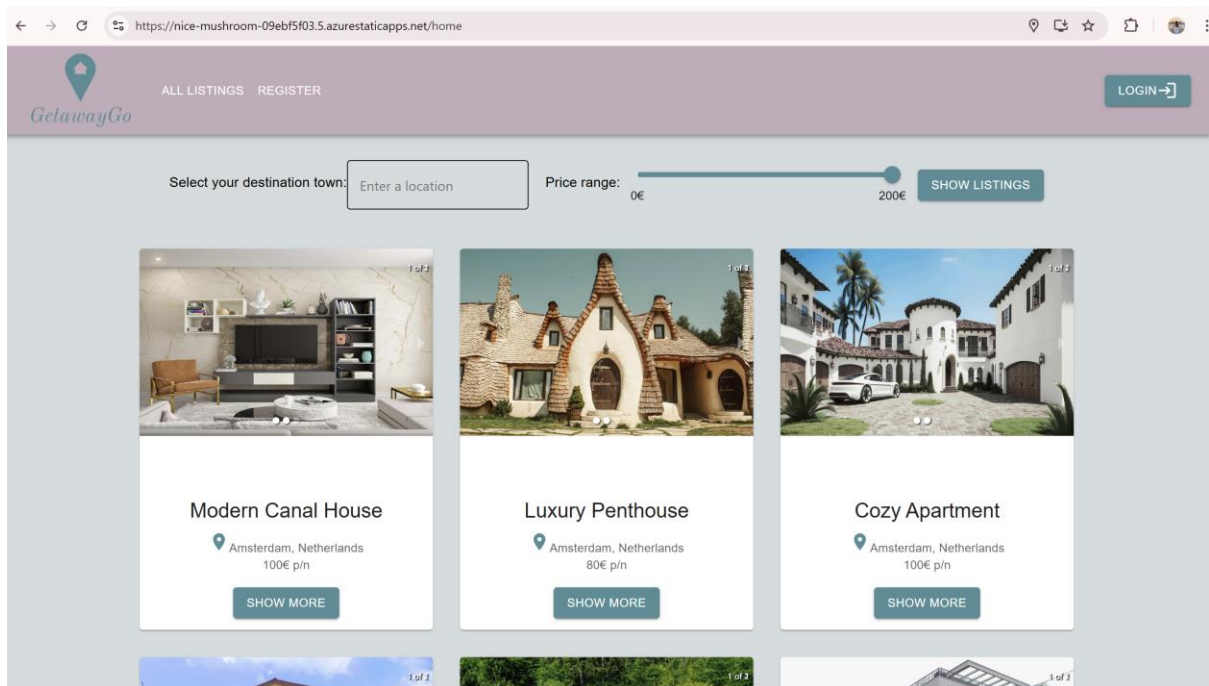


Figure 12 – Website design

Maintainability

Microservices Architecture

Independent microservices simplify debugging, testing, and deploying updates without affecting the entire system.

Logging and Monitoring

Centralized logging and monitoring provides valuable insights for maintenance.

CI/CD Pipeline Testing

Automated tests, including unit, integration, and end-to-end tests, are run in the pipeline to catch and fix issues early. Additional tools like Snyk and OWASP ZAP are integrated. Approval for deployment is also required.




























Recently run pipelines	
Pipeline	Last run
 azure-static-web-apps-nice-mushroom-...	#20250119.3 • Updated azure-static-web-apps-nice-mushroom-09ebf5f03...  Manually triggered for  main
 ApiGateway	#20241208.1 • routes update  Individual CI for  main
 NotificationService	#20241208.2 • pipelines  Individual CI for  main
 ReviewService	#20241208.2 • pipelines  Individual CI for  main
 AnalyticsService	#20241208.2 • pipelines  Individual CI for  main
 ChatService	#20241208.3 • fix  Individual CI for  main
 PropertyManagementService	#20241207.11 • bug fix  Individual CI for  main
 BookingService	#20241207.13 • try with all steps  Individual CI for  main
 UserManagementService	#20241103.1 • allow cors  Individual CI for  main

Figure 13 – Overview of all pipelines

▼	✓	Build	1m 30s
	✓	Initialize job	<1s
	✓	Checkout UserMa...	2s
	✓	NuGetToolInstaller	<1s
	✓	NuGetCommand	4s
	✓	VSBuild	5s
	✓	VSTest	7s
	✓	SnykSecurityScan	44s
	✓	Run OWASP ZAP...	16s
	✓	PublishPipelineArt...	5s
	✓	PublishBuildArtifa...	4s
	✓	Post-job: Checko...	<1s
	✓	Finalize Job	<1s

Figure 14 – Steps in a pipeline for a microservice

Jobs			
▼	✓	Build and Deploy Job	6m 0s
	✓	Initialize job	1s
	✓	Checkout Frontend@m...	1s
	✓	Install and Build R...	1m 41s
	✓	Run Cypress Tests	1m 7s
	✓	AzureStaticWebApp	3m 8s
	✓	Post-job: Checkout Fr...	<1s
	✓	Finalize Job	<1s

Figure 15 – Steps in the pipeline for the frontend

Waiting for review

×

Deploy

[View all](#)

🕒

Approval

📄 Environment **Production**

Waiting for approval • "Approve if the service should be deploy..."

⋮

Comment (optional)

Reject

Approve

⌵

Figure 16 – Approval before deployment to Production

Reliability

Disaster Recovery

I have created a Disaster Recovery plan that needs to be followed in case of failures.

Conclusion

GetawayGo has successfully implemented non-functional requirements to deliver a high-quality, secure, scalable, and maintainable system.