QWEST

Individual Project

Software Architecture Document

Date: 03.06.2024

Version: Version 1.3

Version History

Version	Date	Author	Changes	State
1.0	28.03.2024	Claudiu Badea	Initial Document	Complete
1.1	19.04.2024	Claudiu Badea	Updated technology choices and design decisions	Complete
1.2	16.05.2024	Claudiu Badea	Updated Diagrams	Complete
1.3	03.06.2024	Claudiu Badea	Updated Stages	Complete
1.5	21.06.2024	Claudiu Badea	Updated Diagrams	Complete

Contents

1 Introduction

1.1 Purpose

This Software Architecture Document (SAD) offers a detailed overview of the architectural framework for QWEST. It delineates the high-level design choices, structural components, and technologies that underpin the development and functionality of the system.

1.2 Scope

The primary objective of the QWEST is to simplify and personalize the travel planning process. By offering an adaptive and user-friendly platform, it aims to cater to the unique needs and preferences of travellers worldwide, making travel planning an enjoyable and hassle-free experience.

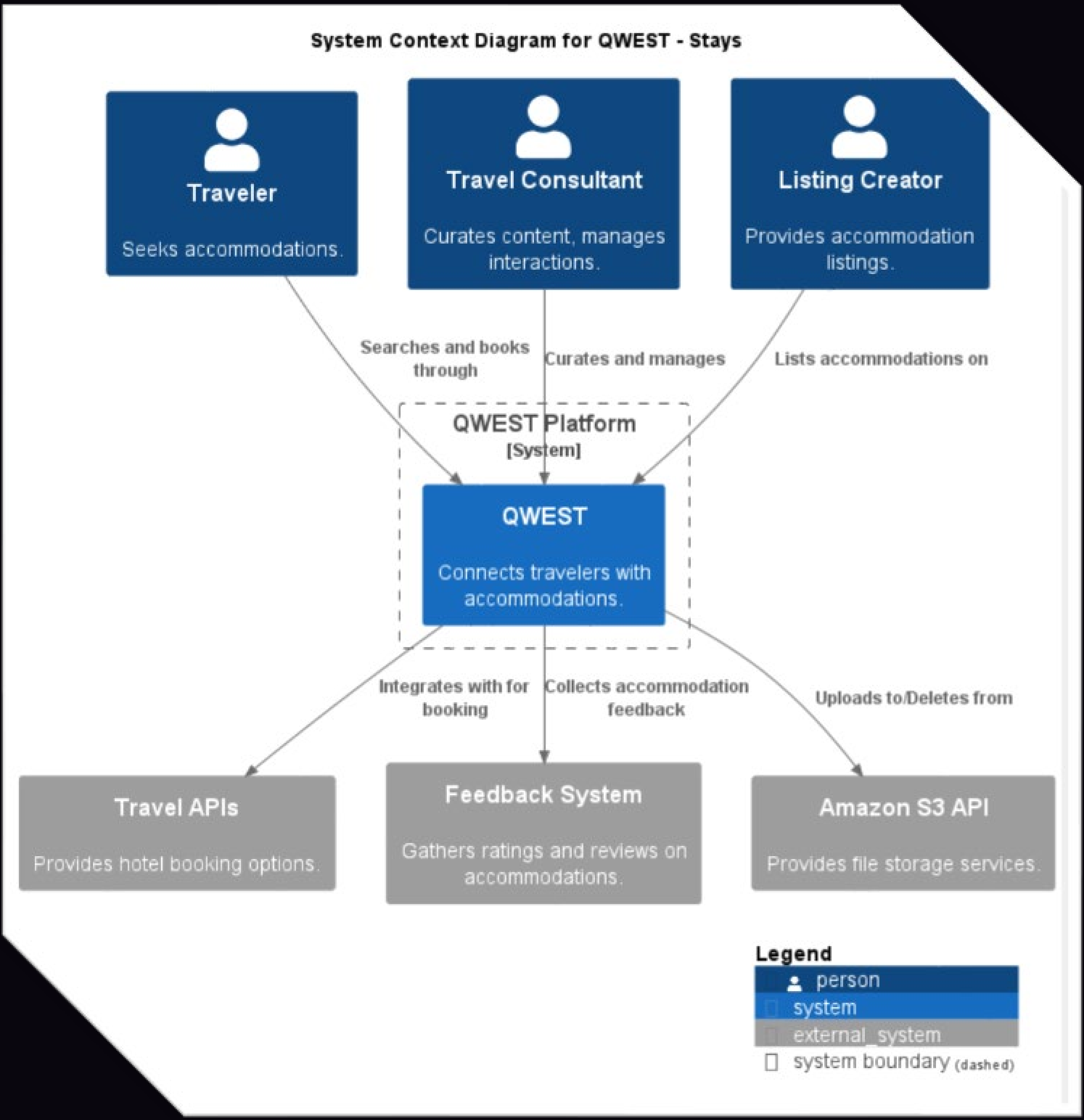
2 System Context

2.1 Business Context

Travelers currently face the challenge of using multiple platforms to plan trips. QWEST aims to unify these aspects into a single, streamlined service, enhancing the user experience by offering a platform that adapts to their personal travel preferences and simplifies the entire planning process.

2.2 System Overview

QWEST consists of a backend powered by Java Spring Boot, which provides RESTful APIs, and a frontend developed with NextJS, facilitating a dynamic and engaging user interface.



System Context Diagram for QWEST - Stays

Traveler
Seeks accommodations.
Travel Consultant
Curates content, manages interactions.
Listing Creator
Provides accommodation listings.
Searches and books through
Curates and manages
Lists accommodations on
QWEST Platform
[System]
QWEST
Connects travelers with accommodations.
Integrates with for booking
Collects accommodation feedback
Uploads to/Deletes from
Travel APIs
Provides hotel booking options.
Feedback System
Gathers ratings and reviews on accommodations.
Amazon S3 API
Provides file storage services.
Legend
person
system
external_system
system boundary (dashed)

3 Containers and Technology Choices

3.1 Backend Container

The contains the core backend infrastructure of QWEST, equipped with powerful API capabilities and server-side processing logic.

Technology Stack:

- Spring Boot: Chosen for its efficiency in development speed, automatic configuration abilities, and comprehensive support within the Spring ecosystem, which is perfect for building microservices.
- RESTful API: Provides a stateless, consistent interface that simplifies the integration and interaction between the frontend and backend elements.

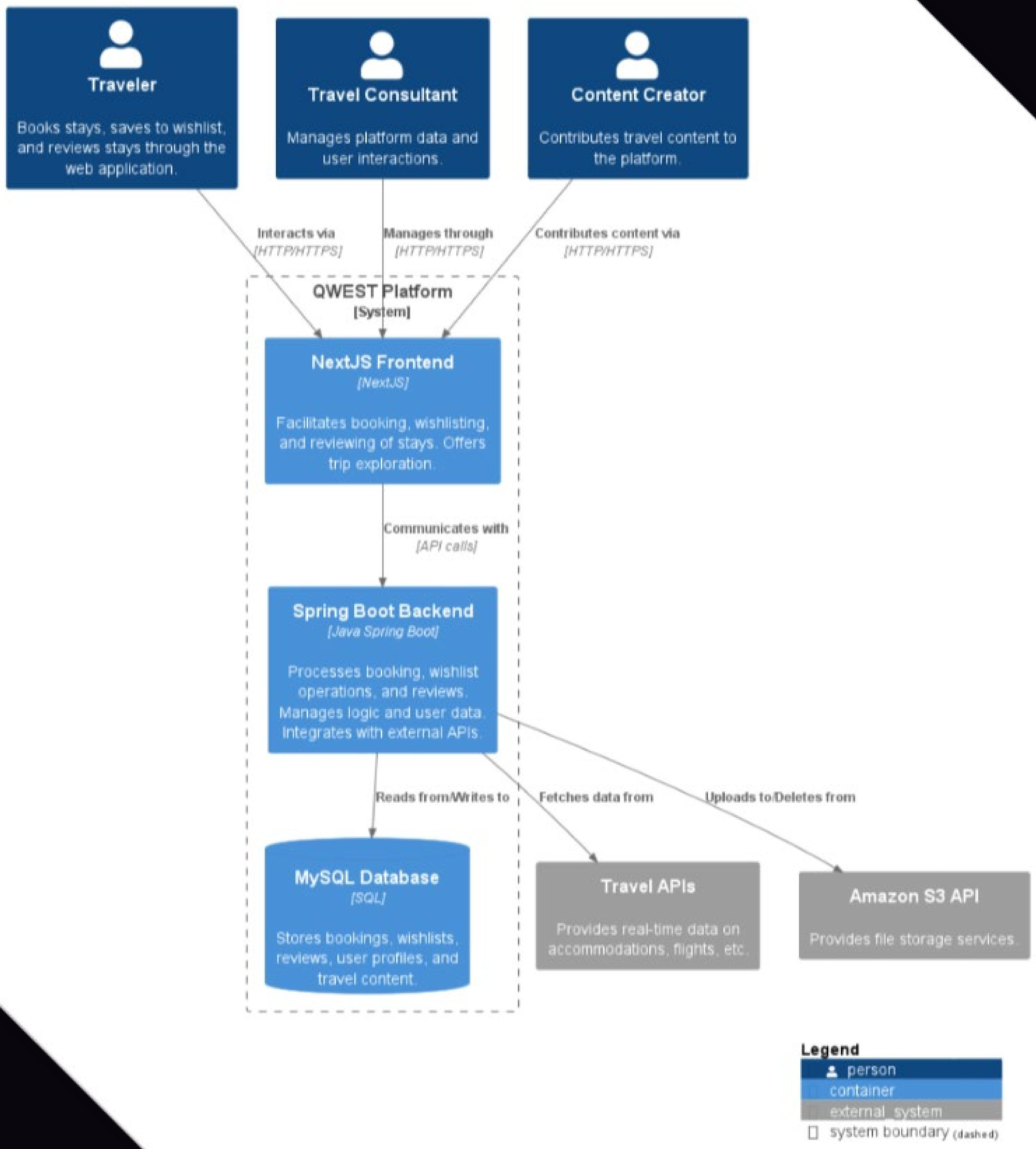
3.2 Frontend Container

The contains the core frontend infrastructure of QWEST, which enables a seamless and responsive user experience.

Technology Stack:

- Next.js: Adopted for its capabilities in server-side rendering and generating static websites, crucial for enhancing the platform's performance and SEO. Its automatic routing system and support for React's component-based architecture also streamline development, making it ideal for building interactive web applications.

Container Diagram for QWEST - Enhanced Traveler Interactions


Traveler
Books stays, saves to wishlist, and reviews stays through the web application.
Travel Consultant
Manages platform data and user interactions.
Content Creator
Contributes travel content to the platform.
Interacts via
[HTTP/HTTPS]
Manages through
[HTTP/HTTPS]
Contributes content via
[HTTP/HTTPS]
QWEST Platform
[System]
NextJS Frontend
[NextJS]
Facilitates booking, wishlisting, and reviewing of stays. Offers trip exploration.
Communicates with
[API calls]
Spring Boot Backend
[Java Spring Boot]
Processes booking, wishlist operations, and reviews. Manages logic and user data. Integrates with external APIs.
Reads from/Writes to
Fetches data from
Uploads to/Deletes from
MySQL Database
[SQL]
Stores bookings, wishlists, reviews, user profiles, and travel content.
Travel APIs
Provides real-time data on accommodations, flights, etc.
Amazon S3 API
Provides file storage services.
Legend
person
container
external_system
system boundary (dashed)

4 Components

4.1 Backend Components

The backend framework is structured into three principal layers: Persistence, Business, and Controller, detailed as follows:

Persistence: Oversees data storage and access, working with the MySQL database to guarantee effective and secure data management. Elements:

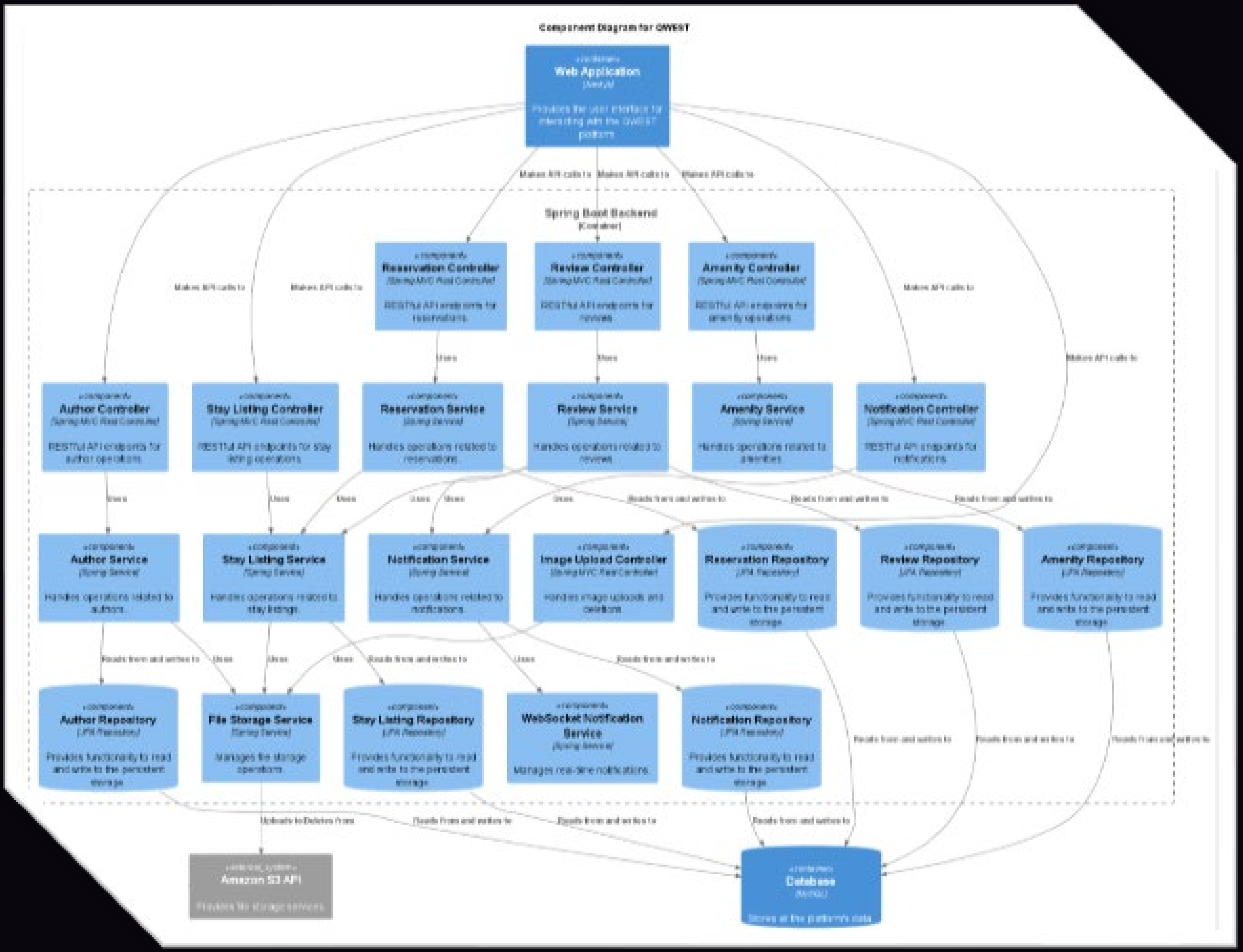
- Entity Classes: Correspond to MySQL tables, serving as data models.
- Repositories: Utilize JpaRepository for object-relational mapping, streamlining database operations.

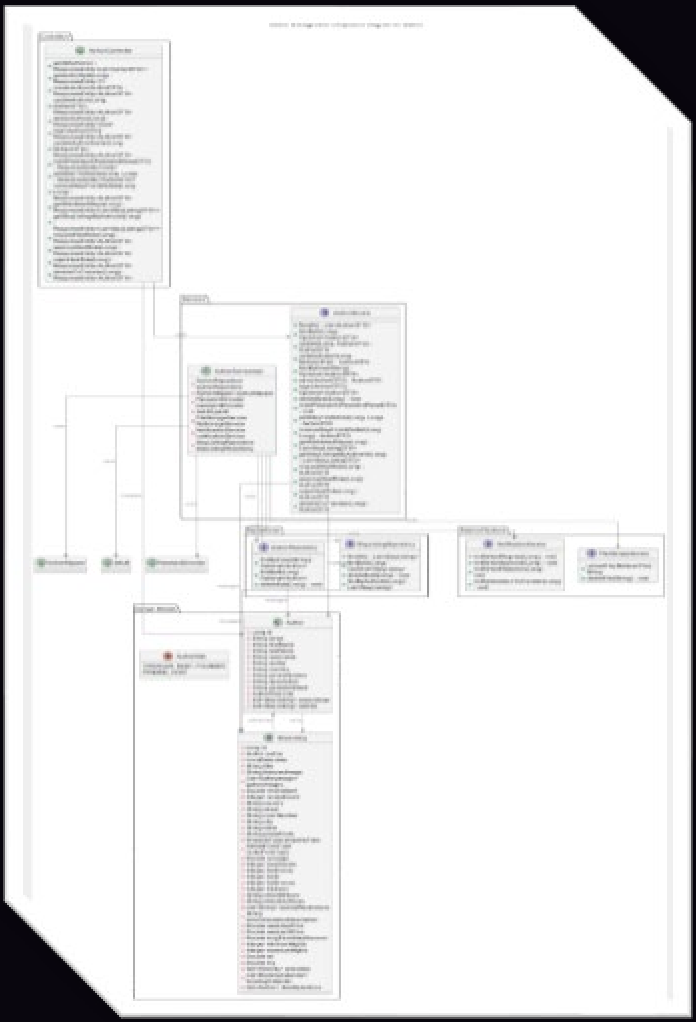
Business: Encapsulates the application's primary logic, modifying data received from the Persistence layer for application consumption.

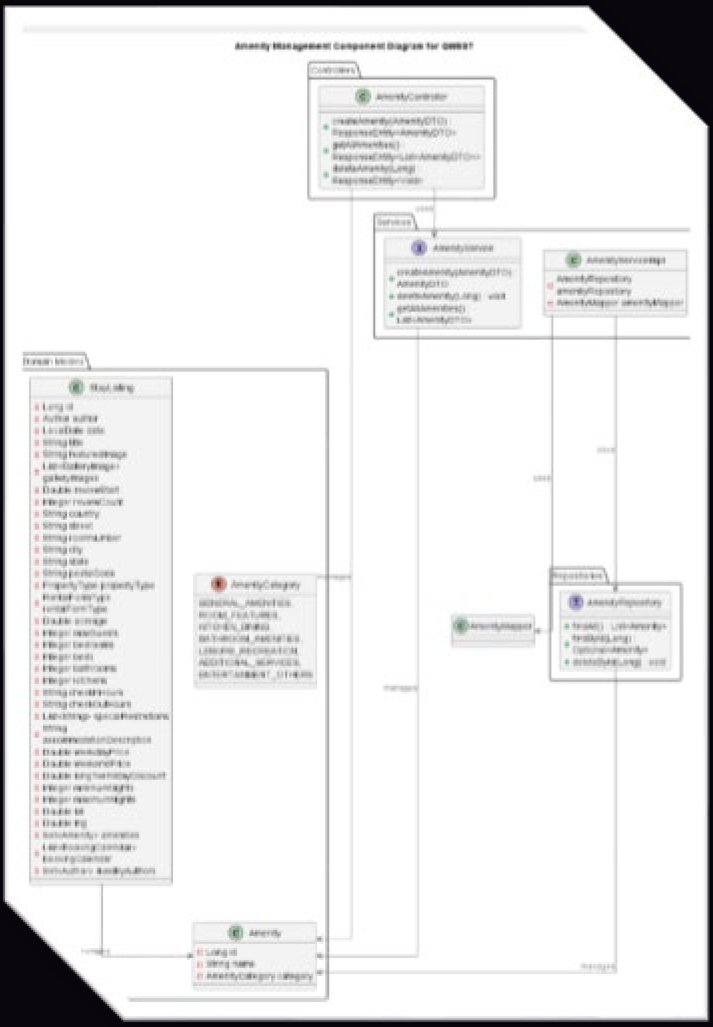
- Service Classes: Embed business logic, preparing data for the Controller layer.
- DTOs (Data Transfer Objects): Support in-application data movement, embodying the YAGNI (You Ain't Gonna Need It) principle by omitting superfluous base classes.

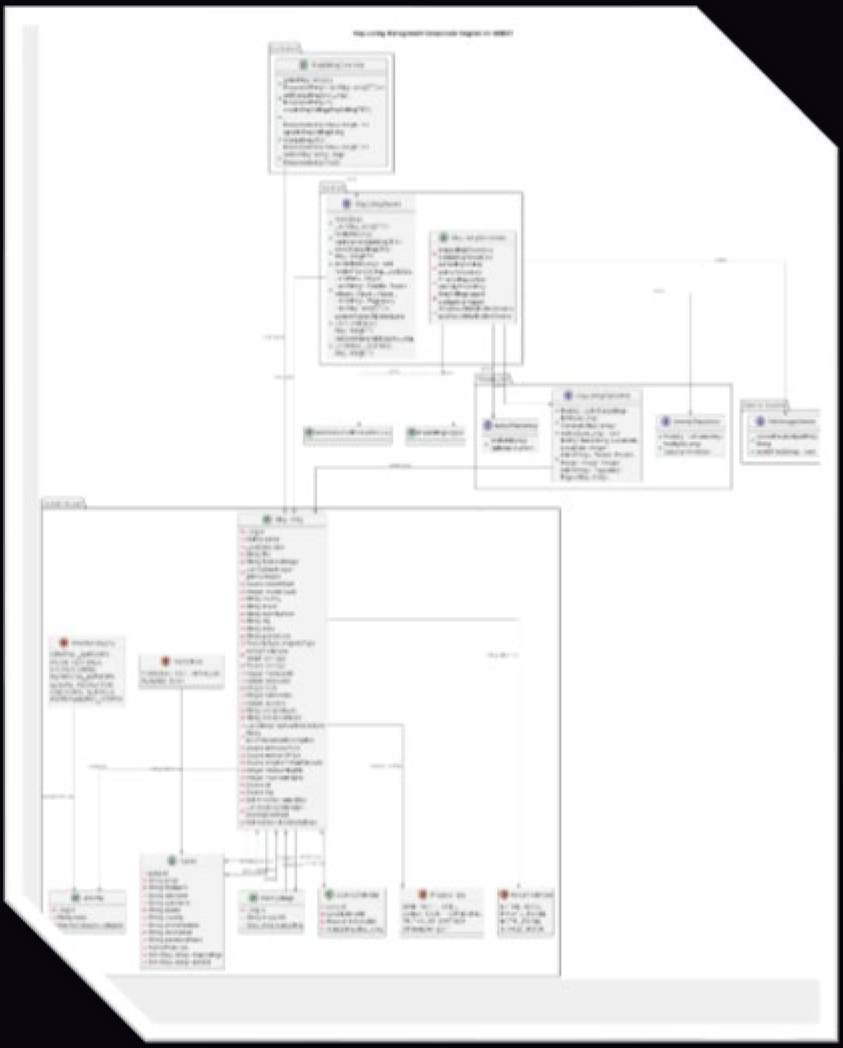
Controller: Directs the flow of data between the user interface and the business logic, interpreting user inputs and delivering suitable responses.

- Controllers: Connect the business layer with the frontend, directing the application’s response to user interactions.


Component Diagram for QWEST
«container»
Web Application
[React]
Provides the user interface for interacting with the QWEST platform
Makes API calls to
Spring Boot Backend
[Container]
«component»
Reservation Controller
[Spring MVC Rest Controller]
RESTful API endpoints for reservations.
«component»
Review Controller
[Spring MVC Rest Controller]
RESTful API endpoints for reviews.
«component»
Amenity Controller
[Spring MVC Rest Controller]
RESTful API endpoints for amenity operations.
Uses
«component»
Author Controller
[Spring MVC Rest Controller]
RESTful API endpoints for author operations.
«component»
Stay Listing Controller
[Spring MVC Rest Controller]
RESTful API endpoints for stay listing operations.
«component»
Reservation Service
[Spring Service]
Handles operations related to reservations.
«component»
Review Service
[Spring Service]
Handles operations related to reviews.
«component»
Amenity Service
[Spring Service]
Handles operations related to amenities.
«component»
Notification Controller
[Spring MVC Rest Controller]
RESTful API endpoints for notifications.
Reads from and writes to
«component»
Author Service
[Spring Service]
Handles operations related to authors.
«component»
Stay Listing Service
[Spring Service]
Handles operations related to stay listings.
«component»
Notification Service
[Spring Service]
Handles operations related to notifications.
«component»
Image Upload Controller
[Spring MVC Rest Controller]
Handles image uploads and deletions
«component»
Reservation Repository
[JPA Repository]
Provides functionality to read and write to the persistent storage
«component»
Review Repository
[JPA Repository]
Provides functionality to read and write to the persistent storage
«component»
Amenity Repository
[JPA Repository]
Provides functionality to read and write to the persistent storage
«component»
Author Repository
[JPA Repository]
Provides functionality to read and write to the persistent storage
«component»
File Storage Service
[Spring Service]
Manages file storage operations.
«component»
Stay Listing Repository
[JPA Repository]
Provides functionality to read and write to the persistent storage
«component»
WebSocket Notification Service
[Spring Service]
Manages real-time notifications.
«component»
Notification Repository
[JPA Repository]
Provides functionality to read and write to the persistent storage
Uploads to/Deletes from
«external_system»
Amazon S3 API
Provides file storage services.
«container»
Database
[MySQL]
Stores all the platform's data


Controllers
AuthorController
Services
AuthorService
Domain Models
Author
StayListing


Amenity Management Component Diagram for QWEST
Controllers
AmenityController
+ createAmenity(AmenityDTO) : ResponseEntity<AmenityDTO>
+ getAllAmenities() : ResponseEntity<List<AmenityDTO>>
+ deleteAmenity(Long) : ResponseEntity<Void>
uses
Services
AmenityService
+ createAmenity(AmenityDTO) : AmenityDTO
+ deleteAmenity(Long) : void
+ getAllAmenities() : List<AmenityDTO>
AmenityServiceImpl
- AmenityRepository amenityRepository
- AmenityMapper amenityMapper
Domain Models
StayListing
- Long id
- Author author
- LocalDate date
- String title
- String featuredImage
- List<GalleryImage> galleryImages
- Double reviewStart
- Integer reviewCount
- String country
- String street
- String roomNumber
- String city
- String state
- String postalCode
- PropertyType propertyType
- RentalFormType rentalFormType
- Double acreage
- Integer maxGuests
- Integer bedrooms
- Integer beds
- Integer bathrooms
- Integer kitchens
- String checkInHours
- String checkOutHours
- List<String> specialRestrictions
- String accommodationDescription
- Double weekdayPrice
- Double weekendPrice
- Double longTermStayDiscount
- Integer minimumNights
- Integer maximumNights
- Double lat
- Double lng
- Set<Amenity> amenities
- List<BookingCalendar> bookingCalendar
- Set<Author> likedByAuthors
AmenityCategory
GENERAL_AMENITIES
ROOM_FEATURES
KITCHEN_DINING
BATHROOM_AMENITIES
LEISURE_RECREATION
ADDITIONAL_SERVICES
ENTERTAINMENT_OTHERS
manages
AmenityMapper
Repositories
AmenityRepository
+ findAll() : List<Amenity>
+ findById(Long) : Optional<Amenity>
+ deleteById(Long) : void
Amenity
- Long id
- String name
- AmenityCategory category



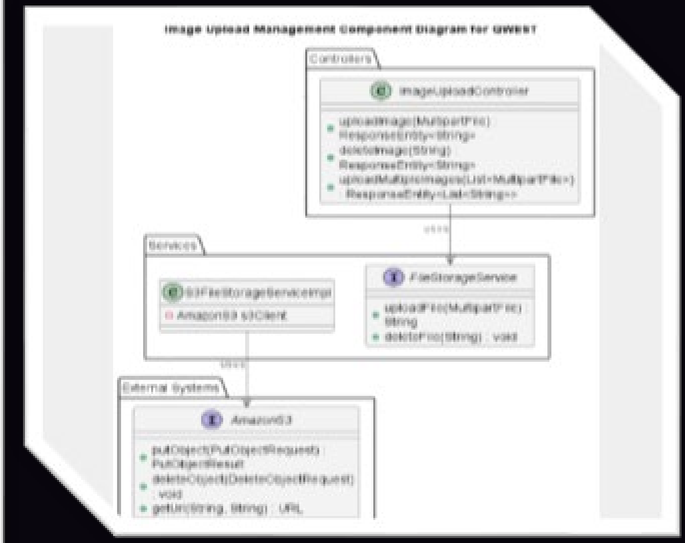
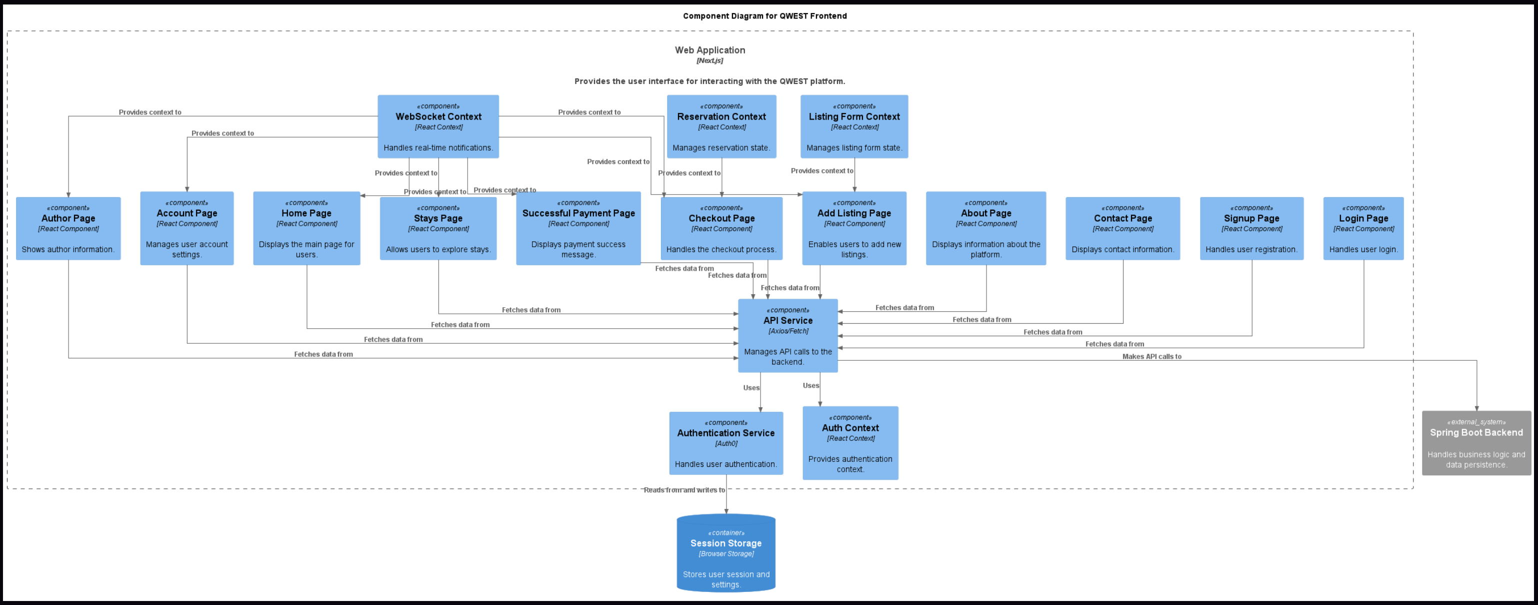


Image Upload Management Component Diagram for QWEST
Controllers
ImageUploadController
uploadImage(MultipartFile)
ResponseEntity<String>
deleteImage(String)
ResponseEntity<String>
uploadMultipleImages(List<MultipartFile>)
ResponseEntity<List<String>>
uses
Services
S3FileStorageServiceImpl
AmazonS3 s3Client
FileStorageService
uploadFile(MultipartFile) : String
deleteFile(String) : void
uses
External Systems
AmazonS3
putObject(PutObjectRequest) : PutObjectResult
deleteObject(DeleteObjectRequest) : void
getUrl(String, String) : URL


Component Diagram for QWEST Frontend
Web Application
[Next.js]
Provides the user interface for interacting with the QWEST platform.
«component»
WebSocket Context
[React Context]
Handles real-time notifications.
«component»
Reservation Context
[React Context]
Manages reservation state.
«component»
Listing Form Context
[React Context]
Manages listing form state.
Provides context to
Provides context to
Provides context to
Provides context to
Provides context to
Provides context to
Provides context to
Provides context to
Provides context to
«component»
Author Page
[React Component]
Shows author information.
«component»
Account Page
[React Component]
Manages user account settings.
«component»
Home Page
[React Component]
Displays the main page for users.
«component»
Stays Page
[React Component]
Allows users to explore stays.
«component»
Successful Payment Page
[React Component]
Displays payment success message.
«component»
Checkout Page
[React Component]
Handles the checkout process.
«component»
Add Listing Page
[React Component]
Enables users to add new listings.
«component»
About Page
[React Component]
Displays information about the platform.
«component»
Contact Page
[React Component]
Displays contact information.
«component»
Signup Page
[React Component]
Handles user registration.
«component»
Login Page
[React Component]
Handles user login.
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
Fetches data from
«component»
API Service
[Axios/Fetch]
Manages API calls to the backend.
Makes API calls to
Uses
Uses
«component»
Authentication Service
[Auth0]
Handles user authentication.
«component»
Auth Context
[React Context]
Provides authentication context.
«external_system»
Spring Boot Backend
Handles business logic and data persistence.
Reads from and writes to
«container»
Session Storage
[Browser Storage]
Stores user session and settings.

4.1 Frontend Components

API

- Manages backend API requests, using HTTP methods for server communication.

Components

- Acts as modular elements for the user interface, enabling the development of dynamic and engaging web pages.

Pages

- Delivers the application's diverse visual representations, employing components to present information and interact with users.

5 CI Pipeline

Continuous Integration (CI) Pipeline Stages

This pipeline is structured into two main phases: build and test.

Build Phase

- Utilizes Gradle to compile and assemble the project components.

Test Phase

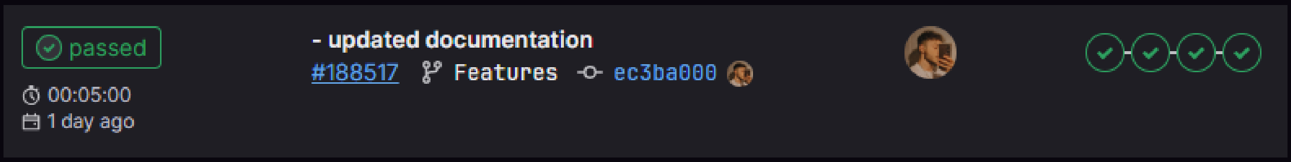
- Employs Gradle again, this time to execute the project's tests, ensuring functionality and reliability.

Sonar Check Phase

- Executes Gradle tasks (“test” and “jacocoTestReport”) and then triggers SonarQube analysis to ensure code quality and security standards are met.

Deploy Stage

- **Setup Docker Environment**: Utilize Docker to configure the necessary environment for deployment.
- **Build Docker Images**: Create Docker images for both backend and frontend services using their respective Dockerfiles.
- **Deploy Services**: Employ docker-compose to deploy the application, ensuring all services are operational. This involves starting containers, setting up networks, and making sure the application is accessible.



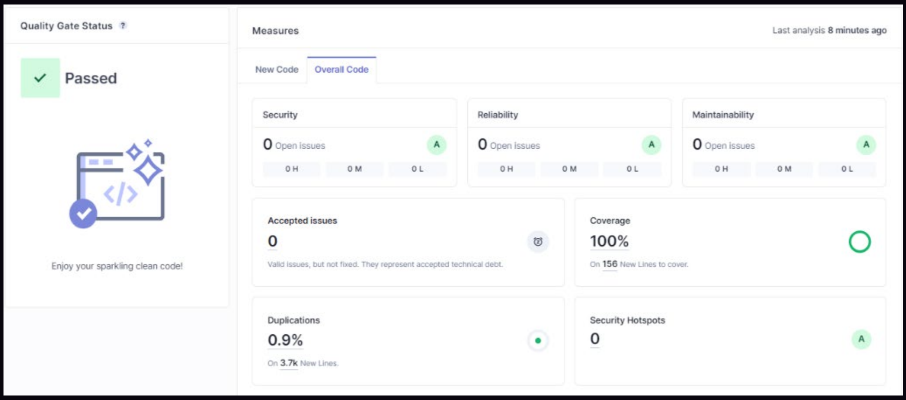
passed

00:05:00

1 day ago

- updated documentation

#188517 Features ec3ba000



Quality Gate Status

Passed

Enjoy your sparkling clean code!

Measures

Last analysis **8 minutes ago**

New Code | Overall Code

Security

0 Open issues A

0 H | 0 M | 0 L

Reliability

0 Open issues A

0 H | 0 M | 0 L

Maintainability

0 Open issues A

0 H | 0 M | 0 L

Accepted issues

0

Valid issues, but not fixed. They represent accepted technical debt.

Coverage

100%

On 156 New Lines to cover.

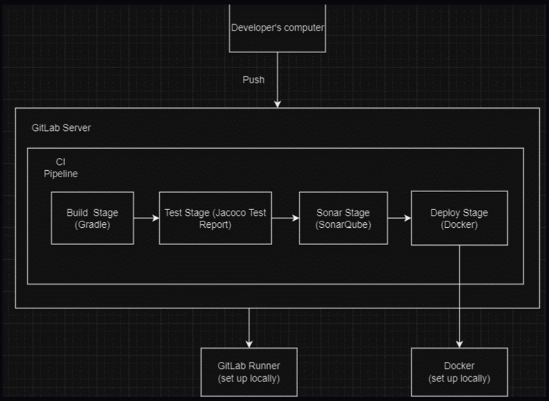
Duplications

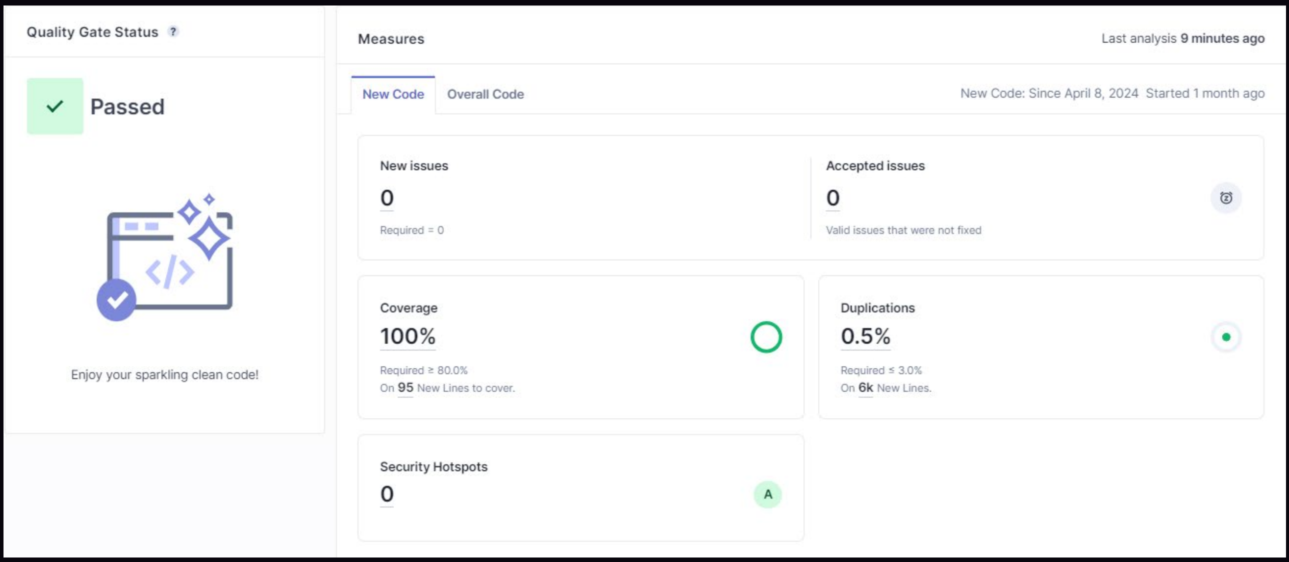
0.9%

On 3.7k New Lines.

Security Hotspots

0 A





Quality Gate Status

Passed

Enjoy your sparkling clean code!

Measures

Last analysis **9 minutes ago**

New Code | Overall Code

New Code: Since April 8, 2024 Started 1 month ago

New issues

0

Required = 0

Accepted issues

0

Valid issues that were not fixed

Coverage

100%

Required ≥ 80.0%

On 95 New Lines to cover.

Duplications

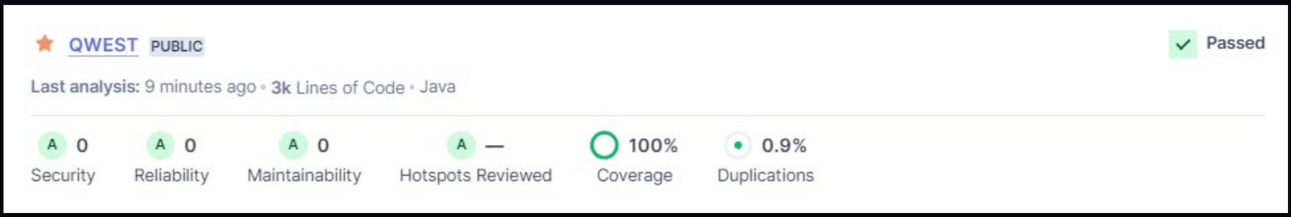
0.5%

Required ≤ 3.0%

On 6k New Lines.

Security Hotspots

0 A



QWEST PUBLIC

Passed

Last analysis: 9 minutes ago • 3k Lines of Code • Java

A 0	A 0	A 0	A —	100%	0.9%
Security	Reliability	Maintainability	Hotspots Reviewed	Coverage	Duplications

6 Security and Authentication

Detailed mechanisms for securing the application and managing user authentication, including the use of Spring Security for robust, stateless API security.

7 Conclusion

This Software Architecture Document outlines the foundational architecture, design motivations, and technical foundations of the QWEST project. Through the adoption of methodologies such as SOLID and YAGNI, and the use of cutting-edge, effective technologies like Spring Boot and Next.js, QWEST is designed to provide a scalable, sustainable, and engaging platform for travelers.