VELOX

MARITO PROJECT: ARCHITECTURAL SPECIFICATION

VERSION: 2.0

MARITO FOR DSFSI

Team Member	Student Number
Arnaud Strydom	u23536013
Hayley Dodkins	u21528790
Zainab Abdulrasaq	u22566202
Palesa Thabane	u21540838
Eugen Vosloo	u20445696

TEAM CONTACT

velox@gmail.com

CONTENTS PAGE

1. INTRODUCTION

THEMARITO PROJECTIS A PLATFORM ENGINEERED TO PROVIDE A SCALABLE, PERFORMANT, AND RELIABLE SYSTEM FOR MANAGING MULTILINGUAL TERMINOLOGY. THIS DOCUMENT SERVES AS THE FORMAL ARCHITECTURAL SPECIFICATION, DETAILING THE TECHNICAL DECISIONS, PATTERNS, AND STRATEGIES THAT FORM THE FOUNDATION OF THE MARITO SYSTEM.

1.1 DOCUMENTPURPOSE

THE PURPOSE OF THIS DOCUMENT IS TO PROVIDE A COMPREHENSIVE ARCHITECTURAL OVERVIEW FOR ALL STAKEHOLDERS, INCLUDING DEVELOPERS, SYSTEM ADMINISTRATORS, AND PROJECT MANAGERS. IT CONNECTS THE "WHY" OF THE PROJECT'S REQUIREMENTS TO THE "HOW" OF ITS TECHNICAL IMPLEMENTATION, ENSURING ALIGNMENT AND A SHARED UNDERSTANDING OF THE SYSTEM'S DESIGN.

1.2 SCOPE

THIS SPECIFICATION COVERS THE SYSTEM'S ARCHITECTURAL STRUCTURE, QUALITY ATTRIBUTES, DEPLOYMENT MODEL, AND TECHNOLOGY STACK. IT DETAILS THE HIGH-LEVEL PATTERNS AND THE LOW-LEVEL DESIGN PATTERNS THAT GOVERN THE IMPLEMENTATION OF THE FRONTEND, BACKEND, AND DATA PERSISTENCE LAYERS.

2. ARCHITECTURAL DESIGN STRATEGY

THE ENTIRE SYSTEM DESIGN WAS SHAPED BY A FEW CRUCIAL NON-FUNCTIONAL REQUIREMENTS (NFRS). THIS REQUIREMENTS-DRIVEN APPROACH WAS CHOSEN TO ENSURE THAT THE FOUNDATIONAL ARCHITECTURE DIRECTLY AND VERIFIABLY SUPPORTS THE PROJECT'S MOST CRITICAL QUALITY ATTRIBUTES. BY PRIORITIZING NFRS SUCH AS SCALABILITY, OFFLINE CAPABILITY, AND SECURITY FROM THE OUTSET, THE PROJECT COULD MAKE INFORMED TRADE-OFFS. FOR INSTANCE, THE SELECTION OF A MICROSERVICES ARCHITECTURE WAS A DIRECT CONSEQUENCE OF PRIORITIZING SCALABILITY AND MAINTAINABILITY OVER THE INITIAL DEVELOPMENT SIMPLICITY THAT A MONOLITH MIGHT OFFER. THIS STRATEGY ENSURES THE FINAL PRODUCT IS ROBUST, EFFICIENT, AND FIT FOR PURPOSE.

3. ARCHITECTURAL DESIGN AND PATTERNS

3.1. HIGH-LEVEL ARCHITECTURAL STYLES

- THE STRUCTURE OF THE MARITO SYSTEM IS BASED ON A COMBINATION OF MODERN ARCHITECTURAL STYLES AND PATTERNS, CHOSEN SPECIFICALLY TO MEET ITS CORE REQUIREMENTS.

- **CLIENT-SERVER ARCHITECTURE:** THIS IS THE FOUNDATIONAL PATTERN FOR THE ENTIRE SYSTEM. IT LOGICALLY SEPARATES THE USER INTERFACE (THE CLIENT) FROM THE BUSINESS LOGIC AND DATA STORAGE (THE SERVER). THIS SEPARATION WAS CRUCIAL FOR ALLOWING INDEPENDENT DEVELOPMENT AND DEPLOYMENT OF THE FRONTEND AND BACKEND, AND IT ENABLES THE CLIENT TO BE AN INTELLIGENT, APPLICATION-LIKE PWA RATHER THAN JUST A THIN DISPLAY LAYER.

- **MICROSERVICES ARCHITECTURE:** THIS PATTERN STRUCTURES THE BACKEND AS A COLLECTION OF SMALL, INDEPENDENT SERVICES, WHERE EACH SERVICE IS RESPONSIBLE FOR A SPECIFIC BUSINESS CAPABILITY (E.G., AUTHENTICATION, SEARCH, VOTING). THIS STYLE WAS CHOSEN BECAUSE IT DIRECTLY ADDRESSES THE REQUIREMENTS FOR SCALABILITY AND MAINTAINABILITY. EACH SERVICE CAN BE DEVELOPED, DEPLOYED, AND SCALED INDEPENDENTLY, ALLOWING, FOR EXAMPLE, THE SEARCH-SERVICE TO BE SCALED UP TO HANDLE HIGH TRAFFIC WITHOUT AFFECTING THE AUTH-SERVICE.

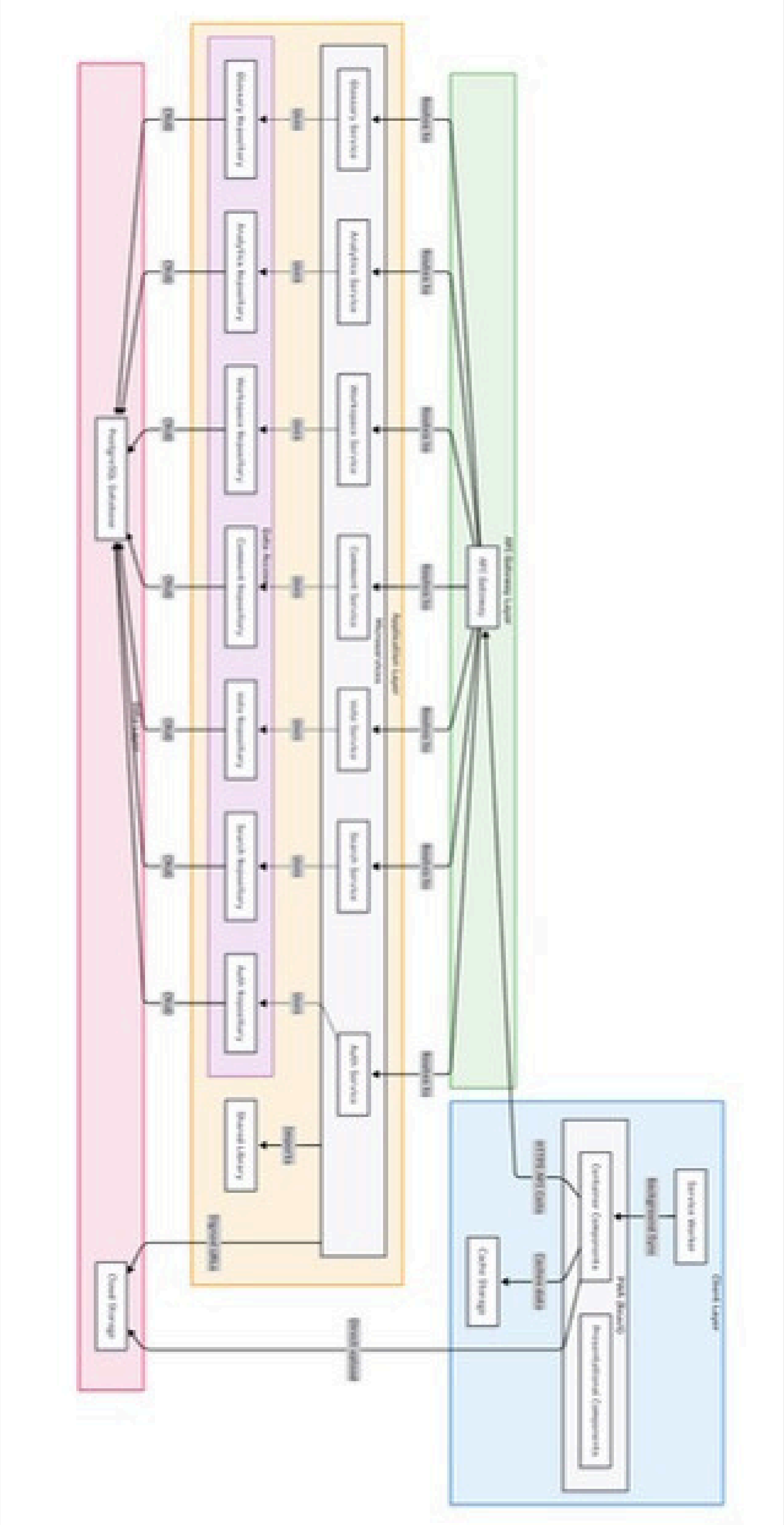
- **COMPONENT-BASED ARCHITECTURE & PWA DESIGN:** THESE FRONTEND PATTERNS WORK TOGETHER TO CREATE A RELIABLE, FAST, AND MAINTAINABLE USER EXPERIENCE THAT FUNCTIONS OFFLINE. THE PWA PATTERN, IMPLEMENTED USING A SERVICE WORKER , WAS SELECTED TO FULFILL THE OFFLINE CAPABILITY REQUIREMENT (NFR3). IT ACHIEVES THIS BY CACHING APPLICATION ASSETS AND USING WORKBOX BACKGROUND SYNC TO QUEUE FAILED POST REQUESTS (LIKE VOTES) AND SEND THEM WHEN THE CONNECTION IS RESTORED. THE COMPONENT-BASED PATTERN, IMPLEMENTED WITH REACT, BREAKS THE UI INTO ENCAPSULATED, REUSABLE PIECES WITH THEIR OWN STATE AND LOGIC (E.G., SEARCHBAR, TERMCARD). THIS IS MORE SPECIFIC THAN A CLASSIC MVC PATTERN AND WAS CHOSEN TO MAKE THE UI HIGHLY MODULAR AND EASIER TO MANAGE AND TEST, DIRECTLY CONTRIBUTING TO A RESPONSIVE AND MAINTAINABLE USER EXPERIENCE (NFR3.1).

- **API GATEWAY:** AS A CRITICAL PART OF THE MICROSERVICES ARCHITECTURE, THE API GATEWAY PROVIDES A SINGLE, UNIFIED ENTRY POINT FOR ALL CLIENT REQUESTS. THIS PATTERN WAS CHOSEN FOR SECURITY (NFR5) AND OPERATIONAL SIMPLICITY. IT HANDLES CROSS-CUTTING CONCERNS LIKE TLS TERMINATION, AUTHENTICATION, AND ROUTING REQUESTS TO THE CORRECT INTERNAL MICROSERVICE, ENSURING THAT NO SERVICE IS EVER DIRECTLY EXPOSED TO THE PUBLIC INTERNET.

3.2. DESIGN PATTERNS

- **REPOSITORY PATTERN :** TO DECOUPLE THE BUSINESS LOGIC FROM THE DATA ACCESS LAYER, A REPOSITORY PATTERN IS EMPLOYED. EACH MICROSERVICE THAT INTERACTS WITH THE POSTGRESQL DATABASE (E.G., SEARCH-SERVICE) HAS A REPOSITORY CLASS RESPONSIBLE FOR ALL DATABASE OPERATIONS (CRUD). THIS ABSTRACTS THE DATA SOURCE, MAKING THE SYSTEM MORE MODULAR AND EASIER TO TEST, AS THE DATABASE CAN BE MOCKED.
- **DEPENDENCY INJECTION :** FASTAPI HAS FIRST-CLASS SUPPORT FOR DEPENDENCY INJECTION. THIS PATTERN IS USED THROUGHOUT THE BACKEND TO MANAGE DEPENDENCIES. A SERVICE LAYER FUNCTION DECLARES A REPOSITORY INSTANCE AS A DEPENDENCY, AND FASTAPI AUTOMATICALLY CREATES AND PROVIDES THAT INSTANCE. THIS PROMOTES LOOSE COUPLING AND ENHANCES TESTABILITY.
- **FACADE PATTERN :** THE API GATEWAY ITSELF IS A PRIME EXAMPLE OF THE FACADE PATTERN. IT PROVIDES A SIMPLE, UNIFIED INTERFACE (A SINGLE API ENDPOINT) TO A MORE COMPLEX UNDERLYING SYSTEM (THE COLLECTION OF DISPARATE MICROSERVICES). THIS SIMPLIFIES INTERACTION FOR THE CLIENT AND DECOUPLES IT FROM THE INTERNAL STRUCTURE OF THE BACKEND.

3.3 ARCHITECTURE DIAGRAM



4. QUALITY REQUIREMENTS

THEFOLLOWING QUALITY REQUIREMENTS, WHICH ARE SPECIFIED IN A TESTABLE WAY, ARE THE MOST IMPORTANT DRIVERS FOR THE MAVITO PROJECT'S ARCHITECTURE. **4.1. SCALABILITY & PERFORMANCE** THE SYSTEM MUST BE ABLE TO HANDLE A GROWING NUMBER OF USERS AND A LARGE DATASET WITHOUT ANY DEGRADATION IN PERFORMANCE. THIS WAS THE PRIMARY DRIVER FOR CHOOSING A MICROSERVICES ARCHITECTURE.

STIMULUS SOURCE: USER AND DATA GROWTH **STIMULUS:** THE NUMBER OF USERS AND THE SIZE OF THE TERMINOLOGY DATASET INCREASE OVER TIME, PLACING A HEAVY LOAD ON THE SYSTEM. **RESPONSE:**

- THE SYSTEM SCALES HORIZONTALLY TO HANDLE INCREASED REQUEST TRAFFIC BY DEPLOYING SERVICES AS DOCKER CONTAINERS ON GOOGLE CLOUD RUN .
- ASYNCHRONOUS APIS BUILT WITH FASTAPI ENSURE THE SERVER REMAINS RESPONSIVE AND AVOIDS BLOCKING WHILE WAITING FOR DATABASE OPERATIONS .
- THE POSTGRESQL DATABASE LEVERAGES INDEXING ON FREQUENTLY SEARCHED COLUMNS TO PROVIDE FAST LOOKUPS FOR LARGE DATASETS.

RESPONSE MEASURE: API RESPONSE TIMES WILL REMAIN UNDER 250MS FOR 95% OF REQUESTS UNDER NORMAL LOAD. THE SYSTEM WILL AUTOMATICALLY SCALE TO HANDLE UP TO 80 CONCURRENT USER REQUESTS WITHOUT PERFORMANCE DEGRADATION PER INSTANCE WITH AN LIMIT OF 100 INSTANCES. DATABASE QUERIES FOR INDEXED TERMS WILL EXECUTE IN UNDER 300MS. **ENVIRONMENT:** THE SYSTEM IS UNDER NORMAL AND PEAK LOAD CONDITIONS. **ARTIFACT:** THE BACKEND SERVICES (GOOGLE CLOUD RUN), API FRAMEWORK (FASTAP WITH API GATEWAY), AND DATABASE (GOOGLE CLOUD SQL).

4.2. USABILITY & OFFLINE ACCESSIBILITY

THEAPPLICATION MUST BE FAST, RESPONSIVE,AND FUNCTIONAL EVEN WHEN USERS HAVE POOR OR NO NETWORK CONNECTIVITY. THIS REQUIREMENT WAS THE DIRECT DRIVER FOR CHOOSING A PROGRESSIVE WEB APP (PWA) ARCHITECTURE.

STIMULUS SOURCE: A USER ACCESSING THE APPLICATION.

STIMULUS: A USER ATTEMPTS TO LOAD THE APPLICATION OR PERFORM AN ACTION (LIKE VOTING) IN A LOW-BANDWIDTH OR OFFLINE ENVIRONMENT.

RESPONSE:

- THE APPLICATION SHELL LOADS INSTANTLY FROM A LOCAL CACHE ON SUBSEQUENT VISITS USING A SERVICE WORKER . THE UI IS BUILT WITH MODULAR,
- REUSABLE COMPONENTS USING REACT FOR A RESPONSIVE EXPERIENCE . ACTIONS PERFORMED OFFLINE ARE SAVED IN A BROWSER QUEUE USING WORKBOX
- BACKGROUND SYNC AND ARE AUTOMATICALLY SENT WHEN CONNECTIVITY IS RESTORED.

RESPONSE MEASURE: THE APPLICATION ACHIEVES A GOOGLE LIGHTHOUSE PERFORMANCE SCORE OF 90+.

- THE APPLICATION IS FULLY FUNCTIONAL OFFLINE FOR ALL CACHED RESOURCES.
- OFFLINE DATA SYNCHRONIZATION COMPLETES SUCCESSFULLY WITHIN 5 MINUTES OF A STABLE CONNECTION BEING RESTORED .

ENVIRONMENT: A USER'S BROWSER, POTENTIALLY WITH AN UNRELIABLE OR NONEXISTENT NETWORK CONNECTION.
ARTIFACT: THE FRONTEND APPLICATION, SPECIFICALLY ITS PWA IMPLEMENTATION USING SERVICE WORKERS AND ITS COMPONENT-BASED UI BUILT WITH REACT.

4.3. AVAILABILITY

THESYSTEMMUSTBE HIGHLY RELIABLE AND ACCESSIBLE TO USERS.

STIMULUS SOURCE: AN INTERNAL MONITORING SERVICE OR AN EXTERNAL USER.
STIMULUS: A HARDWARE FAILURE OR SERVICE CRASH OCCURS IN ONE OF THE SYSTEM'S COMPONENTS.
RESPONSE:

- GOOGLE CLOUD RUN, AS A MANAGED SERVICE, AUTOMATICALLY ENSURES THAT SERVICES ARE RELIABLE AND RESTARTED ON FAILURE .
- THE DATABASE IS CONFIGURED FOR HIGH AVAILABILITY (HA) ON GOOGLE CLOUD SQL, WHICH MANAGES REPLICATION AND FAILOVER AUTOMATICALLY.

RESPONSE MEASURE:

- THE SYSTEM WILL ACHIEVE AT LEAST 99.5% UPTIME, EXCLUDING SCHEDULED MAINTENANCE.
- AUTOMATIC FAILOVER FOR THE DATABASE COMPLETES IN UNDER 5 MINUTES.

ENVIRONMENT: THE PRODUCTION CLOUD ENVIRONMENT.
ARTIFACT: THE MANAGED SERVICES FOR THE BACKEND (GOOGLE CLOUD RUN) AND DATABASE (GOOGLE CLOUD SQL WITH HA CONFIGURATION).

4.4. SECURITY

ALLUSERDATA ANDSYSTEM FUNCTIONS MUST BE PROTECTED FROM UNAUTHORIZED ACCESS AND POTENTIAL THREATS.

STIMULUS SOURCE: AN INTERNAL OR EXTERNAL ACTOR ATTEMPTING TO ACCESS SYSTEM RESOURCES.

STIMULUS: AN ATTEMPT IS MADE TO ACCESS A PROTECTED BACKEND SERVICE DIRECTLY OR TO UPLOAD A SENSITIVE DOCUMENT WITHOUT PROPER AUTHENTICATION.

RESPONSE:

- A GOOGLE API GATEWAY ACTS AS A SINGLE, SECURE ENTRY POINT, ENFORCING HTTPS AND PROTECTING INTERNAL SERVICES FROM DIRECT PUBLIC EXPOSURE .
- ROLE-BASED ACCESS CONTROL IS ENFORCED VIA TOKEN-BASED AUTHENTICATION USING JSON WEB TOKENS (JWT) .
- SECURE, TEMPORARY SIGNED URLS ARE GENERATED FOR DIRECT FILE UPLOADS TO A PRIVATE GOOGLE CLOUD STORAGE BUCKET.

RESPONSE MEASURE:

- ALL PROTECTED API ENDPOINTS REQUIRE A VALID JWT.
- THE SYSTEM PASSES MONTHLY VULNERABILITY SCANS AGAINST THE OWASP TOP 10.
- NO MICROSERVICE IS DIRECTLY ACCESSIBLE FROM THE PUBLIC INTERNET; ALL TRAFFIC IS ROUTED THROUGH THE API GATEWAY.

ENVIRONMENT: THE SYSTEM IS OPERATING AND EXPOSED TO THE PUBLIC INTERNET VIA A SINGLE GATEWAY.

ARTIFACT: THE GOOGLE API GATEWAY, AUTHENTICATION SERVICE (AUTH-SERVICE), AND THE FILE UPLOAD MECHANISM USING GCS SIGNED URLS.

4.5. MAINTAINABILITY & EXTENSIBILITY

THEPROJECT CODEBASE MUST BE EASYFOR DEVELOPERS TO UNDERSTAND, MODIFY, AND EXTEND WITH NEW FEATURES.

STIMULUS SOURCE: A DEVELOPER ON THE PROJECT TEAM.

STIMULUS: A DEVELOPER NEEDS TO DEPLOY A NEW FEATURE, UPDATE THE DATABASE SCHEMA, OR FIX A BUG IN A SPECIFIC PART OF THE SYSTEM.

RESPONSE:

- THE BACKEND IS SPLIT INTO INDEPENDENT MICROSERVICES, ALLOWING FOR FOCUSED DEVELOPMENT AND DEPLOYMENT WITHOUT AFFECTING OTHER TEAMS .
- THE ENTIRE BACKEND IS CONTAINERIZED WITH DOCKER, ENSURING CONSISTENT ENVIRONMENTS .
- DATABASE MIGRATIONS ARE MANAGED THROUGH CODE WITH ALEMBIC FOR SAFE, REPEATABLE, AND VERSION-CONTROLLED SCHEMA CHANGES .
- A FULLY AUTOMATED CI/CD PIPELINE USING GITHUB ACTIONS HANDLES TESTING AND DEPLOYMENT.

RESPONSE MEASURE

- THE CI/CD PIPELINE SUCCESSFULLY VALIDATES AND DEPLOYS A CHANGE TO PRODUCTION IN UNDER 15 MINUTES .
- A NEW DEVELOPER CAN SET UP A LOCAL DEVELOPMENT ENVIRONMENT AND RUN THE SYSTEM USING DOCKER IN UNDER 1 HOUR.
- CODE QUALITY CHECKS ARE ENFORCED AUTOMATICALLY ON EVERY COMMIT VIA PRE-COMMIT HOOKS.

ENVIRONMENT: THE DEVELOPMENT, TESTING, AND PRODUCTION ENVIRONMENTS.
ARTIFACT: THE MICROSERVICES ARCHITECTURE, DOCKER CONTAINERS, ALEMBIC FOR DATABASE MIGRATIONS, AND THE GITHUB ACTIONS CI/CD PIPELINE.

5. DEPLOYMENT MODEL

THISSECTION DESCRIBESHOWTHE MARITO SYSTEM IS DEPLOYED, SUPPORTING ITS CORE REQUIREMENTS FOR SCALABILITY, RELIABILITY, AND MAINTAINABILITY.

THE SYSTEM IS DEPLOYED IN A CLOUD-NATIVE ENVIRONMENT ON GOOGLE CLOUD PLATFORM (GCP). THE DEPLOYMENT TOPOLOGY CONSISTS OF CONTAINERIZED MICROSERVICES RUNNING ON A SERVERLESS PLATFORM, FRONTED BY A SECURE API GATEWAY, WITH A MANAGED DATABASE AND A STATICALLY-HOSTED FRONTEND.

TARGET ENVIRONMENT: THE ENTIRE SYSTEM RUNS ON GOOGLE CLOUD PLATFORM (GCP). THIS WAS CHOSEN TO LEVERAGE MANAGED SERVICES THAT REDUCE OPERATIONAL OVERHEAD AND SUPPORT THE QUALITY REQUIREMENTS.

DEPLOYMENT TOPOLOGY: THE ARCHITECTURE IS A MULTI-TIER, DISTRIBUTED SYSTEM:

FRONTEND: THE REACT-BASED PROGRESSIVE WEB APP IS DEPLOYED AS STATIC ASSETS TO GITHUB PAGES, WHICH PROVIDES GLOBAL CONTENT DELIVERY AND HIGH AVAILABILITY FOR THE USER INTERFACE.

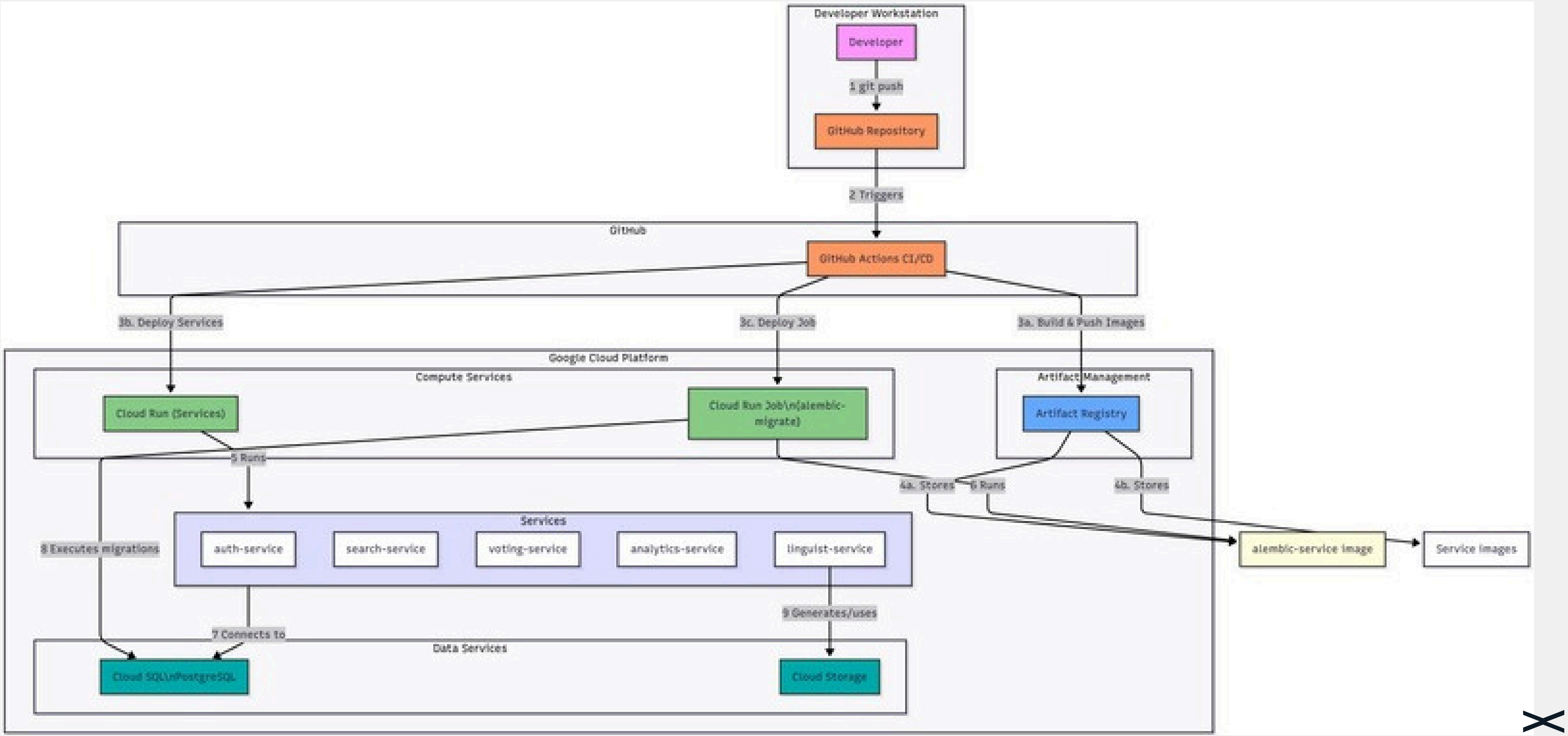
BACKEND: THE BACKEND IS COMPOSED OF MULTIPLE MICROSERVICES (E.G., AUTH-SERVICE, SEARCH-SERVICE). EACH SERVICE IS PACKAGED AS A DOCKER CONTAINER TO ENSURE CONSISTENCY AND PORTABILITY. THESE CONTAINERS ARE ARE PUSHED TO AN ARTIFACT REGISTRY AND THEN DEPLOYED TO GOOGLE CLOUD RUN FROM THE REGISTRY, A SERVERLESS PLATFORM THAT AUTOMATICALLY MANAGES SCALING, FROM ZERO TO A 100 INSTANCES, BASED ON REQUEST TRAFFIC. THIS DIRECTLY SUPPORTS THE SCALABILITY REQUIREMENT.

DATABASE: THE POSTGRESQL DATABASE IS HOSTED ON GOOGLE CLOUD SQL, A FULLY MANAGED SERVICE. FOR RELIABILITY, IT IS CONFIGURED FOR HIGH AVAILABILITY (HA) TO ENSURE AUTOMATIC REPLICATION AND FAILOVER.

GATEWAY: A GOOGLE API GATEWAY SERVES AS THE SINGLE ENTRY POINT FOR ALL FRONTEND REQUESTS, ROUTING THEM TO THE APPROPRIATE INTERNAL MICROSERVICE ON CLOUD RUN.

TOOLS AND AUTOMATION: DOCKER IS USED FOR CONTAINERIZING ALL BACKEND SERVICES. GITHUB ACTIONS IS USED FOR THE CI/CD PIPELINE. THIS PIPELINE AUTOMATES TESTING, CODE QUALITY CHECKS, BUILDING DOCKER IMAGES, PUSHING THEM TO GOOGLE ARTIFACT REGISTRY, AND DEPLOYING NEW VERSIONS TO CLOUD RUN. THIS AUTOMATION IS KEY TO ACHIEVING MAINTAINABILITY AND RELIABLE DEPLOYMENTS.

5.1 DEPLOYMENT DIAGRAM



6. TECHNOLOGY CHOICES

THEFOLLOWING TECHNOLOGIES WERE CHOSEN TO IMPLEMENT THE ARCHITECTURAL PATTERNS AND SATISFY THE NON-FUNCTIONAL REQUIREMENTS.

6.1.FRONTEND

CONSIDERATION 1: REACT

OVERVIEW: REACT IS A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES, FOCUSING ON A COMPONENT-BASED ARCHITECTURE.

ADVANTAGES: IT HAS A MASSIVE ECOSYSTEM, STRONG COMMUNITY SUPPORT, AND A DECLARATIVE APPROACH THAT MAKES COMPLEX UIS EASIER TO MANAGE. ITS COMPONENT MODEL PROMOTES REUSABILITY AND MAINTAINABILITY.

DISADVANTAGES: IT IS A LIBRARY, NOT A FULL FRAMEWORK, SO ROUTING AND STATE MANAGEMENT OFTEN REQUIRE ADDITIONAL LIBRARIES, LEADING TO MORE DECISIONS FOR THE DEVELOPMENT TEAM.

CONSIDERATION 2: ANGULAR

OVERVIEW: ANGULAR IS A COMPREHENSIVE, OPINIONATED FRONTEND FRAMEWORK DEVELOPED AND MAINTAINED BY GOOGLE.

ADVANTAGES: IT'S A FULL-FLEDGED PLATFORM WITH BUILT-IN SOLUTIONS FOR ROUTING, STATE MANAGEMENT, AND MORE. ITS USE OF TYPESCRIPT PROVIDES STRONG TYPING OUT OF THE BOX.

DISADVANTAGES: IT HAS A STEEPER LEARNING CURVE AND CAN BE MORE VERBOSE THAN REACT. ITS COMPREHENSIVE NATURE CAN BE OVERKILL FOR SMALLER APPLICATIONS AND MAY LEAD TO PERFORMANCE OVERHEAD IF NOT MANAGED CAREFULLY.

FINAL CHOICE & JUSTIFICATION: REACT WITH VITE

THE UI IS BUILT WITH REACT USING A COMPONENT-BASED PATTERN. THIS CHOICE ALIGNS PERFECTLY WITH THE NEED FOR A HIGHLY MODULAR AND MANAGEABLE UI (NFR6), WHERE EACH PIECE IS AN ENCAPSULATED, REUSABLE COMPONENT WITH ITS OWN VIEW, STATE, AND LOGIC. THE BUILD PROCESS IS HANDLED BY VITE, A MODERN TOOL THAT PRODUCES HIGHLY OPTIMIZED AND SMALL STATIC FILES, RESULTING IN FASTER INITIAL PAGE LOAD TIMES AND A BETTER USER EXPERIENCE, DIRECTLY ADDRESSING PERFORMANCE REQUIREMENTS (NFR2).

CONSIDERATION 1: CHART.JS

OVERVIEW: CHART.JS IS A SIMPLE YET FLEXIBLE OPEN-SOURCE JAVASCRIPT LIBRARY FOR DATA VISUALIZATION.

ADVANTAGES: IT'S LIGHTWEIGHT, VERY EASY TO USE, AND HAS GREAT DOCUMENTATION. IT OFFERS A GOOD VARIETY OF POPULAR CHART TYPES (BAR, LINE, PIE, ETC.) AND IS PERFECT FOR STRAIGHTFORWARD DATA VISUALIZATION TASKS.

DISADVANTAGES: IT IS NOT AS POWERFUL OR CUSTOMIZABLE AS MORE COMPLEX LIBRARIES LIKE D3.JS AND MAY LACK SOME ADVANCED OR NICHE CHART TYPES.

CONSIDERATION 2: D3.JS

OVERVIEW: D3.JS (DATA-DRIVEN DOCUMENTS) IS A POWERFUL AND LOW-LEVEL JAVASCRIPT LIBRARY FOR CREATING COMPLEX, DYNAMIC DATA VISUALIZATIONS.

ADVANTAGES: IT OFFERS UNPARALLELED FLEXIBILITY AND CONTROL, ALLOWING FOR THE CREATION OF VIRTUALLY ANY DATA VISUALIZATION IMAGINABLE.

DISADVANTAGES: IT HAS A VERY STEEP LEARNING CURVE AND REQUIRES MANUAL MANIPULATION OF THE DOM, MAKING IT MORE TIME-CONSUMING FOR STANDARD CHARTS.

FINAL CHOICE & JUSTIFICATION: CHART.JS

FOR THE MARITO PROJECT, WHICH REQUIRES A CLEAR AND EFFECTIVE DISPLAY OF STATISTICS, CHART.JS IS THE IDEAL CHOICE. ITS SIMPLICITY, LIGHTWEIGHT NATURE, AND EASE OF INTEGRATION ALLOW FOR RAPID DEVELOPMENT OF NECESSARY DATA VISUALIZATIONS WITHOUT INTRODUCING UNNECESSARY COMPLEXITY OR PERFORMANCE OVERHEAD. THIS ALIGNS WITH THE PROJECT'S CORE FOCUS ON PERFORMANCE (NFR2) AND USABILITY (NFR3)

6.3. BACKEND

CONSIDERATION 1: FASTAPI

OVERVIEW: FASTAPI IS A MODERN, HIGH-PERFORMANCE WEB FRAMEWORK FOR BUILDING APIS WITH PYTHON, BASED ON STANDARD PYTHON TYPE HINTS.

ADVANTAGES: IT OFFERS AUTOMATIC INTERACTIVE DOCUMENTATION, IS BUILT ON ASGI FOR BEST-IN-CLASS ASYNCHRONOUS CAPABILITIES, AND PROVIDES PERFORMANCE ON PAR WITH NODE.JS AND GO.

DISADVANTAGES: ITS ECOSYSTEM IS NEWER AND SMALLER COMPARED TO MORE ESTABLISHED FRAMEWORKS LIKE DJANGO, MEANING FEWER THIRD-PARTY PLUGINS.

CONSIDERATION 2: DJANGO

OVERVIEW: DJANGO IS A HIGH-LEVEL PYTHON WEB FRAMEWORK THAT ENCOURAGES RAPID DEVELOPMENT AND CLEAN, PRAGMATIC DESIGN.

ADVANTAGES: IT IS HIGHLY MATURE, WITH A MASSIVE ECOSYSTEM AND A "BATTERIES-INCLUDED" PHILOSOPHY THAT PROVIDES BUILT-IN SOLUTIONS FOR ORM, ADMIN PANELS, AND MORE.

DISADVANTAGES: IT IS TRADITIONALLY SYNCHRONOUS (THOUGH ASYNC SUPPORT IS IMPROVING) AND ITS MONOLITHIC NATURE CAN BE LESS SUITABLE FOR A MICROSERVICES ARCHITECTURE. ITS ORM CAN ADD OVERHEAD.

FINAL CHOICE & JUSTIFICATION: FASTAPI

THE BACKEND MICROSERVICES ARE BUILT WITH FASTAPI. ITS NATIVE ASYNC/AWAIT MODEL WAS THE DECIDING FACTOR, AS IT ALLOWS THE SERVER TO HANDLE MANY CONCURRENT REQUESTS EFFICIENTLY, PREVENTING I/O OPERATIONS (LIKE DATABASE CALLS) FROM BLOCKING THE ENTIRE SYSTEM. THIS CHOICE IS CRITICAL FOR ACHIEVING THE HIGH PERFORMANCE AND RESPONSIVENESS (NFR2) REQUIRED BY THE SYSTEM, AS IT DRAMATICALLY INCREASES THROUGHPUT UNDER LOAD.

6.4. DATABASE

CONSIDERATION 1: POSTGRESQL

OVERVIEW: POSTGRESQL IS A POWERFUL, OPEN-SOURCE OBJECT-RELATIONAL DATABASE SYSTEM KNOWN FOR ITS RELIABILITY, FEATURE ROBUSTNESS, AND PERFORMANCE.

ADVANTAGES: IT IS HIGHLY COMPLIANT WITH SQL STANDARDS, SUPPORTS ADVANCED DATA TYPES, AND HAS POWERFUL FEATURES LIKE INDEXING, WHICH IS CRITICAL FOR PERFORMANCE. IT IS WELL-SUITED FOR HANDLING COMPLEX, RELATIONAL DATA AT SCALE.

DISADVANTAGES: IT CAN HAVE A MORE COMPLEX SETUP AND ADMINISTRATION COMPARED TO SIMPLER DATABASES LIKE MYSQL FOR BASIC USE CASES.

CONSIDERATION 2: MONGODB

OVERVIEW: MONGODB IS A POPULAR SOURCE-AVAILABLE CROSS-PLATFORM DOCUMENT-ORIENTED DATABASE PROGRAM (NOSQL).

ADVANTAGES: ITS FLEXIBLE DOCUMENT MODEL IS IDEAL FOR UNSTRUCTURED DATA AND ALLOWS FOR RAPID DEVELOPMENT. IT SCALES OUT HORIZONTALLY VERY WELL.

DISADVANTAGES: IT LACKS THE ACID TRANSACTIONAL GUARANTEES OF RELATIONAL DATABASES AND CAN BE LESS EFFICIENT FOR HIGHLY RELATIONAL DATA.

FINAL CHOICE & JUSTIFICATION: POSTGRESQL ON GOOGLE CLOUD SQL

THE PROJECT MIGRATED FROM AN UNSCALABLE IN-MEMORY JSON FILE TO POSTGRESQL TO HANDLE MILLIONS OF RECORDS AND PROVIDE THE ABILITY TO PERFORM COMPLEX QUERIES ON STRUCTURED TERMINOLOGY DATA. TO ENSURE HIGH PERFORMANCE (NFR1.1), THE SYSTEM HEAVILY LEVERAGES DATABASE INDEXING ON FREQUENTLY SEARCHED COLUMNS, ALLOWING THE ENGINE TO FIND DATA ALMOST INSTANTLY. THE DATABASE IS HOSTED ON GOOGLE CLOUD SQL, A MANAGED SERVICE THAT SIMPLIFIES ADMINISTRATION AND ALLOWS FOR EASY SCALING AND HIGH-AVAILABILITY REPLICATION.

7. TECHNOLOGY STACK SUMMARY

Category	Technology
Frontend Framework	React
Frontend Build Tool	Vite
Report Generation	Chart.js
Backend Framework	FastAPI (Python)
Database	PostgreSQL
Deployment Platform	Google Cloud Run
API Management	Google API Gateway
Containerization	Docker
CI/CD	GitHub Actions
Database Migrations	Alembic
Authentication	JSON Web Tokens (JWT)

8. SERVICE CONTRACT MARITO

Thisdocumentoutlines the service-levelagreementand technical contract for the **Marito Platform**. It defines the expectations, responsibilities, and communication structure between the backend and frontend to ensure consistent, secure, and well-defined interactions.

All endpoints are prefixed with `/api/v1`, and all JSON payloads follow standard HTTP request/response patterns.

Authentication Service

The Marito Authentication Service provides essential identity and access management capabilities for the platform. It includes user registration, login with OAuth2, access token issuance, and authenticated endpoints for retrieving user details or generating signed URLs.

Authentication Endpoints

/api/v1/auth/register

Method:

POST

Description:

Registers a new user.

Example Request:

```
{
  "email": "user@example.com",
 "first_name": "string",
 "last_name": "string",
 "role": "linguist",
 "profile_pic_url": "string",
 "password": "string"
}
```

Possible Response Codes:

- 201 Successful Response
- 422 Validation Error

/api/v1/auth/login

Method:

POST

Description:

OAuth2 compatible token login, get an access token for future requests. The frontend LoginPage.tsx sends 'email' (as username) and 'password'.

Example Request:

```
{
    "grant_type" : "string",
    "username" : "string",
    "password" : "string",
    "scope" : "string",
    "client_id" : "string",
    "client_secret" : "string"
}
```

Possible Response Codes:

- 200 Successful Response
- 422 Validation Error

/api/v1/auth/me

Method:

GET

Description:

Get current logged-in user's details.

Possible Response Codes:

- 200 Successful Response

Uploads Endpoints

/api/v1/uploads/generate-signed-url

Method:

POST

Description:

Generates a temporary, secure URL that allows an authenticated user to upload a file directly to a private Google Cloud Storage bucket.

Example Request:

```
{
  "content_type": "string",
 "filename": "string"
}
```

Possible Response Codes:

- 200 Successful Response
- 422 Validation Error

Admin Endpoints

/api/v1/admin/users/{user_id}/role

Method:

GET

Description:

Updates a user role

Required Parameter:

- user_id: string
- new_role: string

Possible Response Codes:

- 200 Successful Response
- 422 Validation Error

/api/v1/admin/users

Method:

GET

Description:

Fetch all users

Possible Response Codes:

- 200 Successful Response

/api/v1/admin/users/{user_id}/uploads

Method:

GET

Description:

Fetch user uploads

Required Parameter:

- user_id: string

Possible Response Codes:

- 200 Successful Response
- 422 Validation Error

/api/v1/admin/download-url

Method:

GET

Description:

Get signed download url.

Required Parameter:

- `gcs_key`: string

Possible Response Codes:

- 200 Successful Response
- 422 Validation Error

/api/v1/admin/users-with-uploads

Method:

GET

Description:

Fetch users with uploads.

Possible Response Codes:

- 200 Successful Response

Health Endpoints

/

Method:

GET

Description:

Health check endpoint. Used to verify that the Search Service is online and responsive.

Possible Response Codes:

- 200 Successful Response

Search Service

The Search Service isresponsible for querying the glossary database to retrieve relevant terms, suggest matching entries, and support health check functionality. This service powers both precise and fuzzy search across multiple filters including language and domain.

Search Endpoints

/api/v1/search/

Method:

GET

Description:

Searches database for terms matching given parameters.

Parameters:

- `query`: string
- `language`: string
- `domain`: string
- `sort_by`: name or popularity
- `page`: number
- `page_size`: number
- `fuzzy`: boolean

Possible Response Codes:

- 201 Successful Response
- 422 Validation Error

Suggest Endpoints

/api/v1/suggest/

Method:

GET

Description:

Returns search suggestions based on a partial query using fuzzy matching.

Parameters:

- `query`: string

Possible Response Codes:

- 201 Successful Response
- 422 Validation Error

/api/v1/glossary/categories

Method:

GET

Descr iption:

Retrieves all available glossary categories.

Possible Response Codes:

- 200 OK

/api/v1/glossary/categories/{category_name}/terms

Method:

GET

Description:

Fetches all terms associated with a specific category.

Path Parameters:

- `category_name`: string

Possible Response Codes:

- 200 OK

- 422 Validation Error

/api/v1/glossary/terms/{term_id}/translations

Method:

GET

Description:

Fetches all translations for a specific term.

Path Parameters:

- `term_id`: string

Possible Response Codes:

- 200 OK
- 422 Validation Error

/api/v1/glossary/search

Performs a simple term search across all categories.

Query Parameters:

- `query`: string

Possible Response Codes:

- 200 OK
- 422 Validation Error

/api/v1/glossary/search

Method:

POST

Description:

Advanced search with filters (language, domain) and pagination.

Query Parameters:

- `query`: string

- `domain`: string
- `language`: string
- `page`: integer (default: 1≣
- `limit`: integer (default: 10≣

Request Body:

```
{
  "query": "string",
 "domain": "string",
 "language": "string"
}
```

Possible Response Codes:

- 200 OK
- 422 Validation Error

/api/v1/glossary/domains

Method:

GET

Description:

Retrieves all available glossary domains.

Possible Response Codes:

- 200 OK

/api/v1/glossary/languages

Method:

GET

Description:

Retrieves all available glossary languages.

Possible Response Codes:

- 200 OK

/api/v1/glossary/translate

Method:

POST

Description:

Translates a list of terms into one or more target languages.

Query Parameters:

- `source_language`: string (default: English)
- `domain`: string

Request Body:

```
{
  "terms": ["string"],
 "target_languages": ["string"]
}
```

Possible Response Codes:

- 200 OK
- 422 Validation Error

/api/v1/glossary/stats

Method:

GET

Description:

Returns statistics about the glossary.

Possible Response Codes:

- 200 OK

/api/v1/glossary/random

Method:

GET

Description:

Returns one or more random terms.

Query Parameters:

- `count`: integer (default: 1≣

Possible Response Codes:

- 200 OK
- 422 Validation Error

Health Endpoints

/

Method:

Get

Description:

Health check endpoint. Used to verify that the Search Service is online and responsive.

Possible Response Codes:

- 201 Successful Response
- 500 Internal Server Error

Analytics Serivce

The Marito Analytics Service provides descriptive statistics and insights derived from the glossary data. It includes coverage metrics, length distributions, and summary reports for terms, languages, and categories.

Analytics Endpoints

/api/v1/analytics/descriptive

Method:

GET

Description:

Returns all descriptive analytics in a combined legacy format.

Possible Response Codes:

- 200 OK

/api/v1/analytics/descriptive/category-frequency

Method:

GET

Description:

Returns frequency distribution of terms across different categories.

Possible Response Codes:

- 200 OK

/api/v1/analytics/descriptive/language-coverage

Method:

GET

Description:

Returns percentage of filled translations per language.

Possible Response Codes:

- 200 OK

/api/v1/analytics/descriptive/term-length

Method:

GET

Description:

Returns average term lengths per language.

Possible Response Codes:

- 200 OK

/api/v1/analytics/descriptive/definition-length

Method:

GET

Description:

Returns average definition lengths per language.

Possible Response Codes:

- 200 OK

/api/v1/analytics/descriptive/unique-terms

Method:

GET

Description:

Returns the number of unique terms per language.

Possible Response Codes:

- 200 OK

/

Method:

GET

Description:

Health check endpoint.

Possible Response Codes:

- 200 OK

Linguist Service

TheLinguist ApplicationService enables authenticated users to submit documentation for linguist verification. This includes uploading identity documents, CVs, certifications, and optional research papers.

Linguist Endpoints

/api/v1/linguist-applications/

Method:

POST

Description:

Creates a new linguist application for the currently authenticated user.

Request Body:

```
json
CopyEdit
{
  "id_document_url": "string",
 "cv_document_url": "string",
 "certifications_document_url": "string",
 "research_papers_document_url": "string"
}
```

Possible Response Codes:

- 201 Created
- 422 Validation Error

Vote Service

The Vote Service enables users to cast votes on glossary terms. Each term supports upvotes and downvotes per user, and vote counts are recalculated in real-time.

Vote Endpoints

/api/v1/votes/

Method:

POST

Description:

Casts or updates a vote for a specific term by the authenticated user. Sends the updated vote count in the response.

Request Body:

```
{
  "term_id": "3fa85f64≣5717≣4562-b3fc-2c963f66afa6",
 "vote": "upvote"
}
```

Possible Response Codes:

- 200 OK

```
{
  "term_id": "3fa85f64≣5717≣4562-b3fc-2c963f66afa6",
 "upvotes"≣ 0,
 "downvotes"≣ 0,
 "user_vote": "upvote"
}
```

- 422 Validation Error