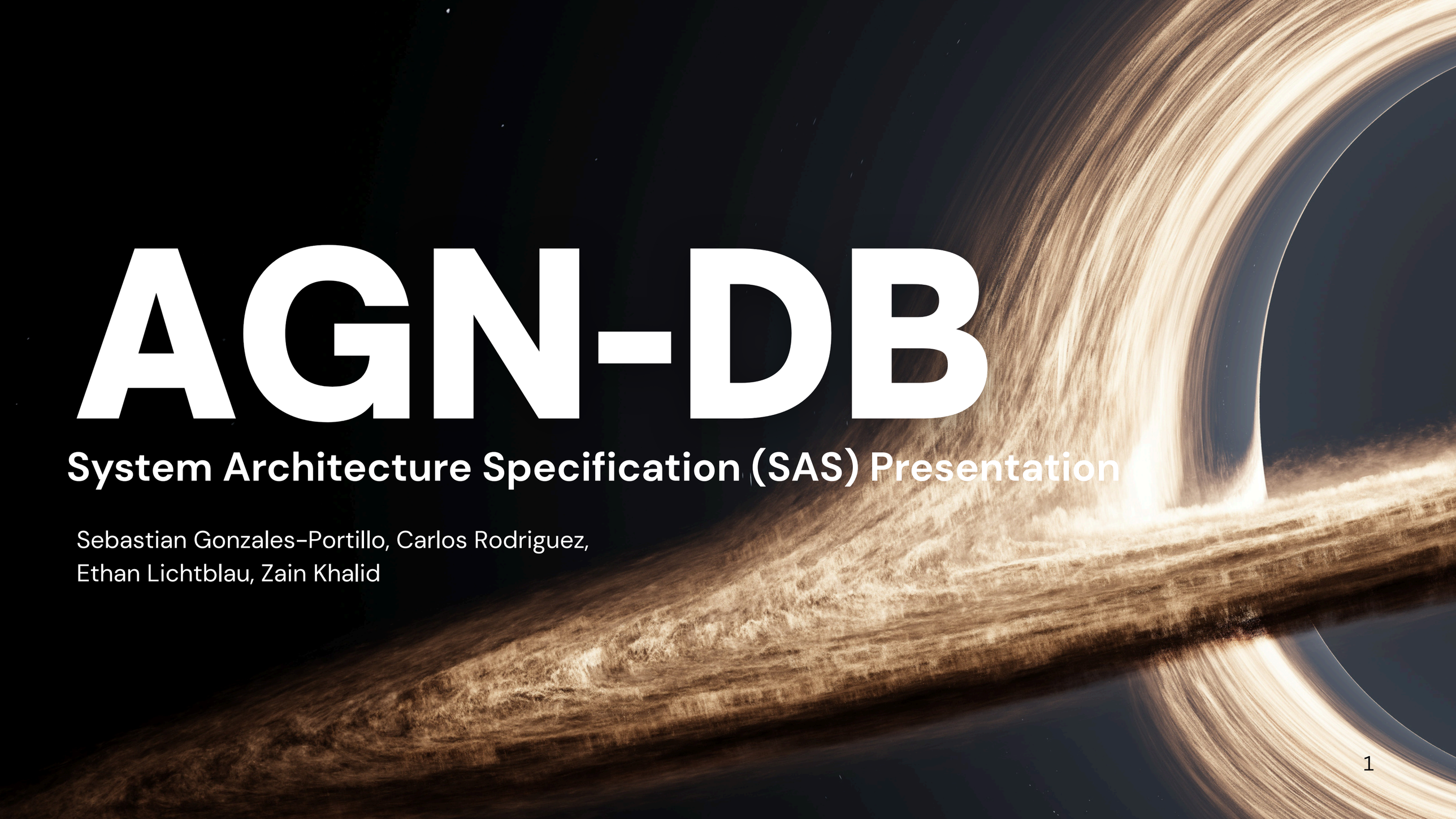


AGN-DB

System Architecture Specification (SAS) Presentation

Sebastian Gonzales-Portillo, Carlos Rodriguez,
Ethan Lichtblau, Zain Khalid

1 - SYSTEM ANALYSIS

1.1 - System Overview

1.2 - System Diagram

1.3 - Actor Idenfifiction

1.4 - Design Rationale

1.1 - SYSTEM OVERVIEW

What is AGN-DB?

- AGN-DB is a sophisticated web application designed for astrophysics research. It provides a platform for exploring, analyzing, and exporting astronomical data.
- Follows a modern microservices architecture with **three main components**:
  - **React-based frontend** web application
  - A **FastAPI**-based REST API backend
  - **MariaDB** database optimized for astronomical data storage
- The architecture will allow for scalability, maintainability, and separation of concerns
  - Ideal for scientific research applications that require both performance and flexibility

1.2 - SYSTEM DIAGRAM

01

Frontend

- React Single-Page Application for User Interaction

02

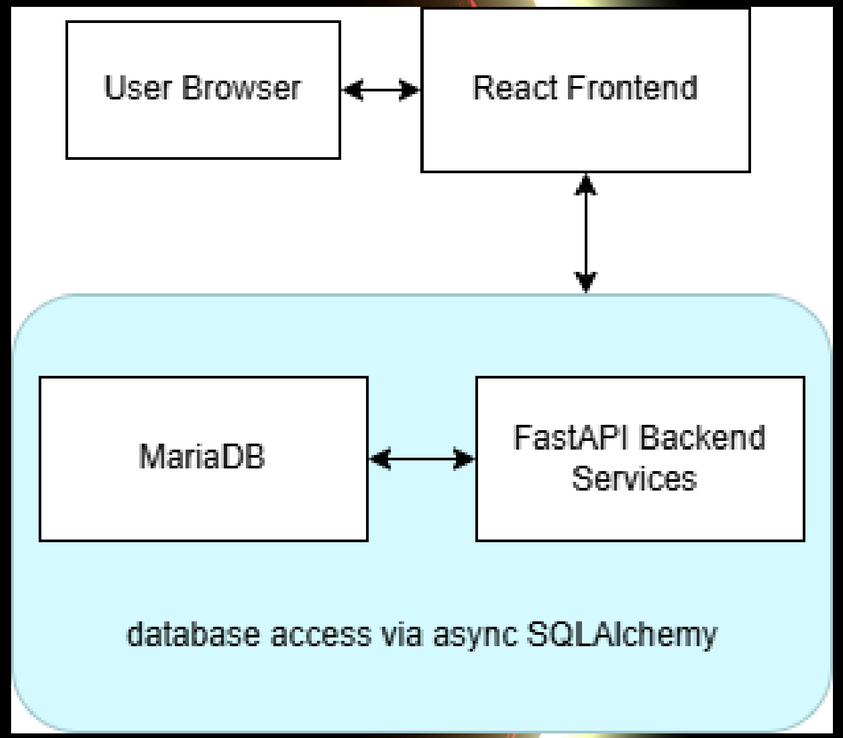
Backend

- FastAPI-based REST API managing business logic and data processing

03

Database

- MariaDB optimized for high-performance astronomical data storage



1.3 - ACTOR IDENTIFICATION

AGN-DB System Actors

- **Astrophysics Researchers**: primary users who search for, analyze, and export Active Galactic Nuclei (AGN) data.
- **Data Administrators:** Manage the system's dataset, handling data imports and quality control.
- **System Administrators:** Maintain the technical infrastructure, handle deployments, and monitor system health.
- **Automated System Services:** Scheduled tasks, data validation services, and external API integrations that interact with the system programmatically.

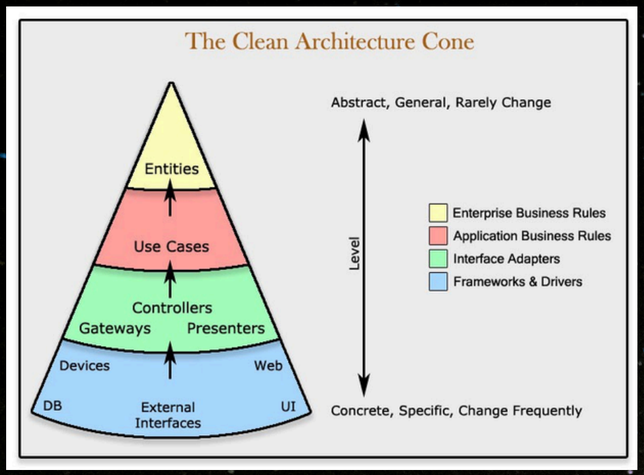
1.4 - DESIGN RATIONALE

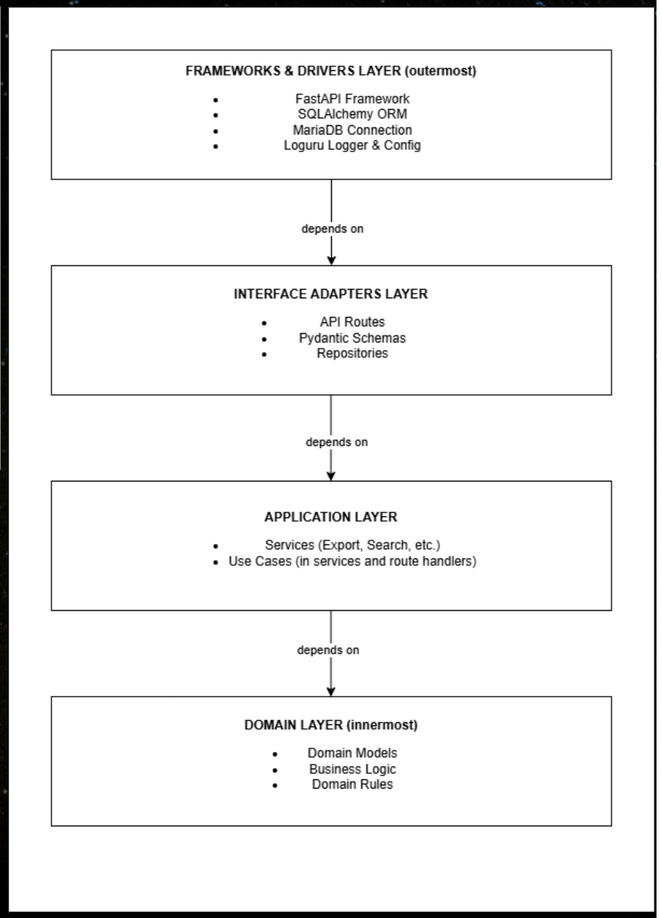
1.4.1 - Architectural Style

1.4.2 - Design Patterns

1.4.3 - Frameworks

1.4.1 - Architectural Style





CLEAN ARCHITECTURE & LAYERED APPROACH

- Organizes system into concentric layers around a central domain model.
- Promotes **separation of concerns** with dependencies pointing inward only.

- Entities (**Domain Layer**): Core business models and rules representing astronomical concepts, independent of application concerns.
- Use Cases (**Application Layer**): Application-specific business logic
- Controllers/Presenters (**Interface Adapters Layer**): Translates between use cases and external frameworks
- Frameworks & Drivers (**Infrastructure Layer**): External technical infrastructure including FastAPI, SQLAlchemy ORM, MariaDB, and logging components.

- Isolates scientific domain from technical concerns.
- Allows independent evolution of APIs and database layers.
- Handles complex astronomical data relationships.
- Improves maintainability and testability over time.

1.4.2 - Design Pattern(s)

REPOSITORY PATTERN

- Abstracts data access logic to create a collection-like interface for domain objects.
- Decouples business logic from storage implementation.

DEPENDENCY INJECTION

- Applied via FastAPI's dependency system to promote loose coupling
- Enhances testability by providing dependencies at runtime.

CQRS-LITE

- Separates read (query) and write (command) operations into distinct paths.
- Allows for independent optimization of each operation type.

1.4.3 - Framework

FASTAPI

- Native *async/await* support for efficient asynchronous operations.
- Auto-generates OpenAPI docs for endpoints
- Validates and serializes requests with Pydantic
- High performance compared to other Python frameworks.
- Strong typing improves dev experience and reliability

SQLALCHEMY

- Works flexibly with raw SQL for complex queries
- Supports asynchronous operations via *asyncmy*
- Robust ORM for domain modeling
- Database-agnostic design allows easy migration
- Strong typing improves dev experience and reliability

MARIADB

- Fork of MySQL
- High performance for 7M+ astrophysical records
- Legacy system – out of project scope for replacement

1.4.3 - Framework

(continued)

REACT

- Component-based UI promotes reusability
- Industry standard with rich community support
- Works seamlessly with TypeScript for type-safe development.

PYDANTIC

- Strong request validation in FastAPI
- Clear error messages for invalid data
- Python-native type annotations

MARIADB

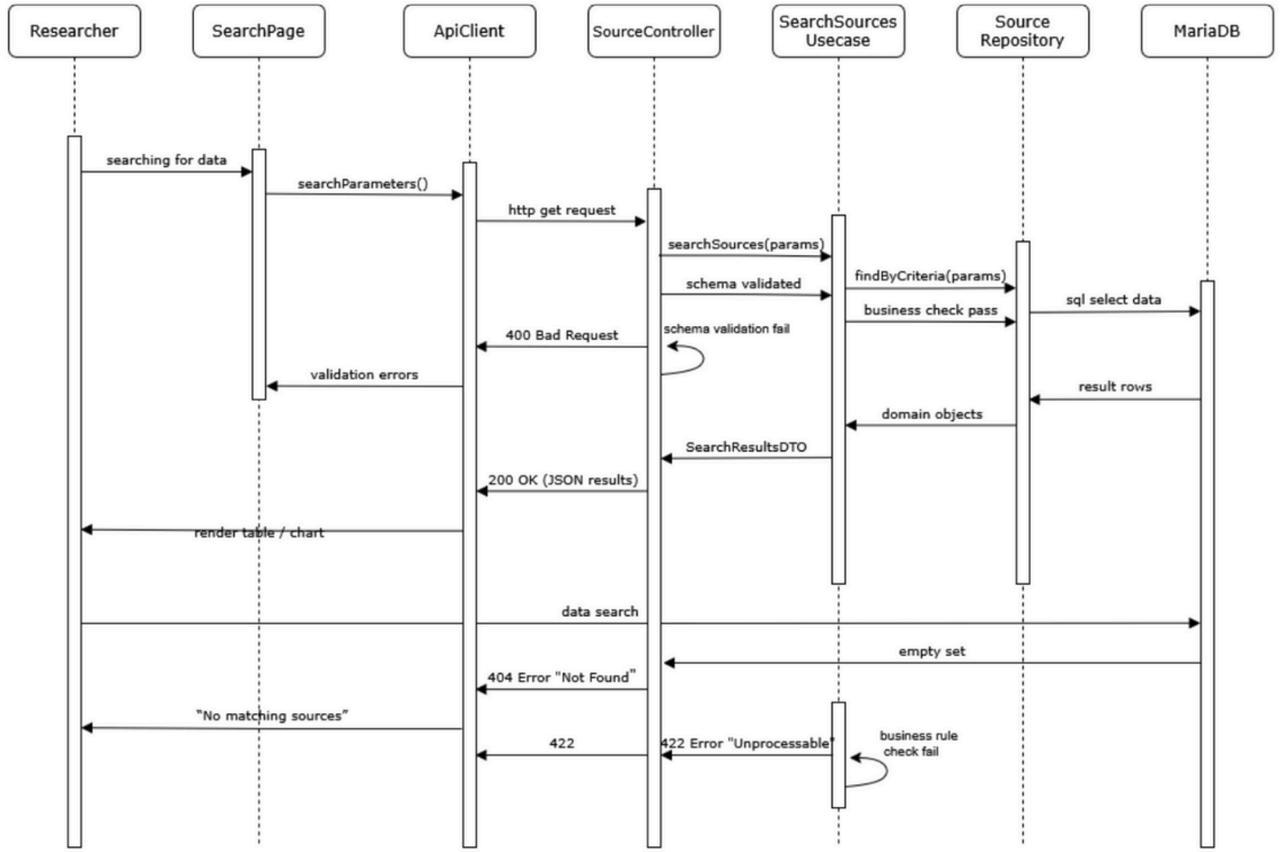
- Easy, structured logging with rich features
- Supports contextual error reporting and exception capture

2 - FUNCTIONAL DESIGN

2.1 – Data Query Workflow

2.2 – Data Export Workflow

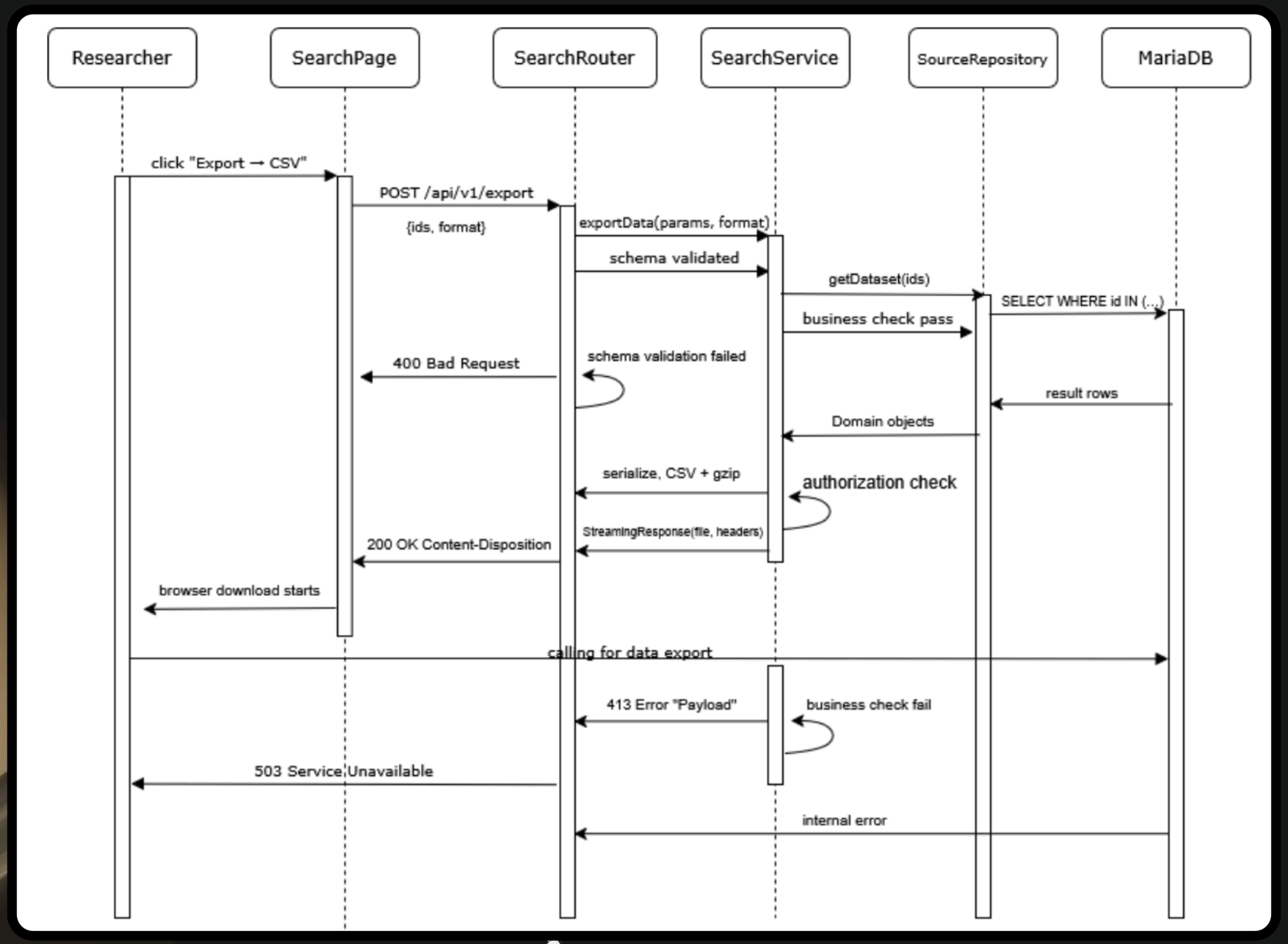
FUNCTIONAL DESIGN 2.1



2.1 – Data Query Workflow

FUNCTIONAL DESIGN 2.2

2.2 – Data Export Workflow



3 - STRUCTURAL DESIGN

3.1 – Core Entity Class Diagram

3.2 – Repository Layer Class Diagram

3.3 – API Class Layer Diagram

3.1 - CORE ENTITY CLASS DIAGRAM

01

Central Entity: SourceAGN

- Represents each astronomical object
- Linked to Photometry, Redshift, and Classification records.

02

One-to-Many Relationships

- Each SourceAGN connects to multiple:
  - Photometry measurements
  - Redshift measurements
  - Classifications

03

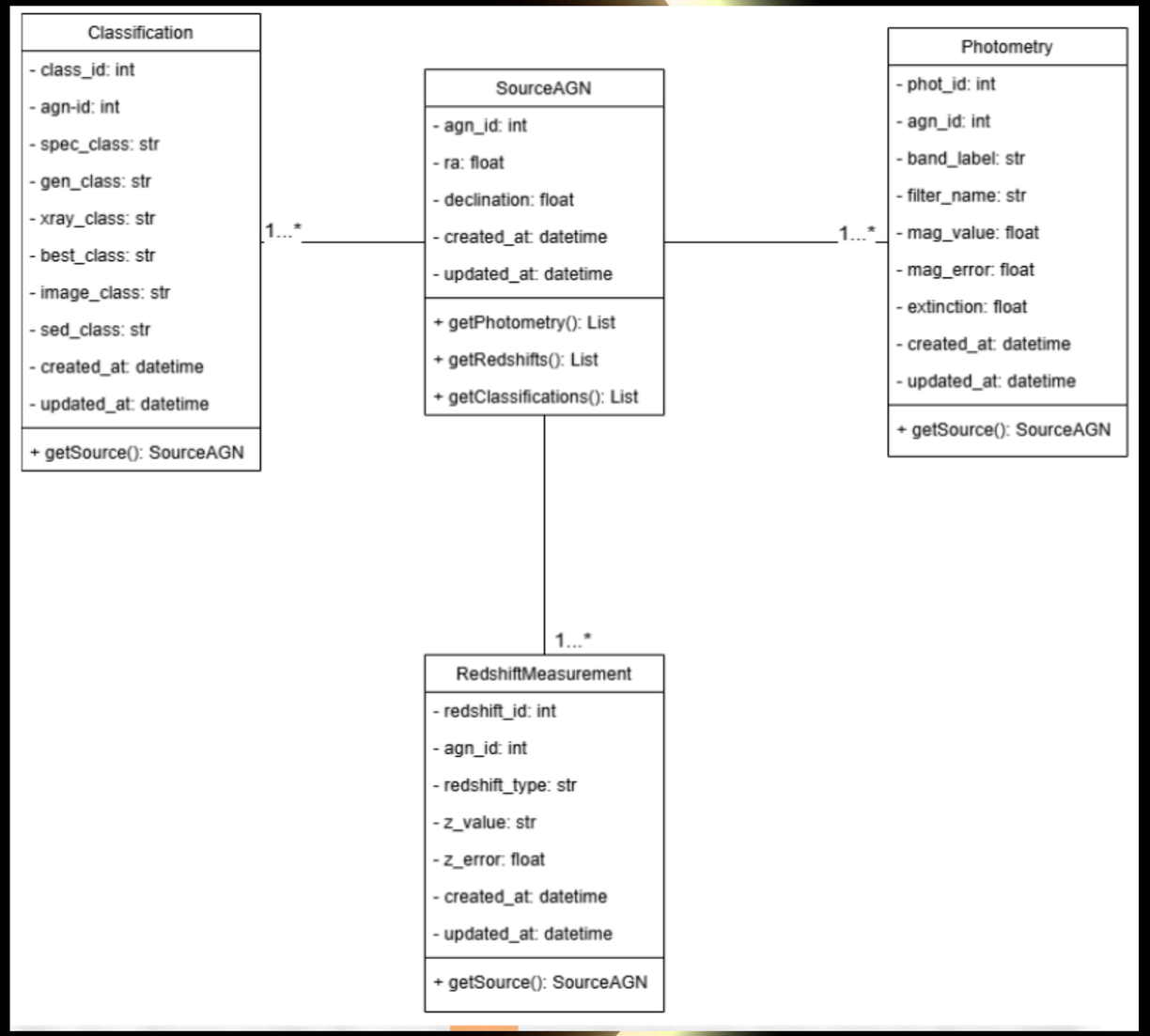
Scientific Data Modeling

- Captures brightness, distance, and type attributes
- Includes timestamps for tracking updates

04

Design Advantages

- Modular & extensible for new data types
- Supports traceability & scientific accuracy



3.2 - REPOSITORY LAYER CLASS DIAGRAM

01

BaseRepository Abstraction

- Defines common async CRUD methods.
- Uses SQLAlchemy sessions for database access.
- All other repositories inherit from this base.

02

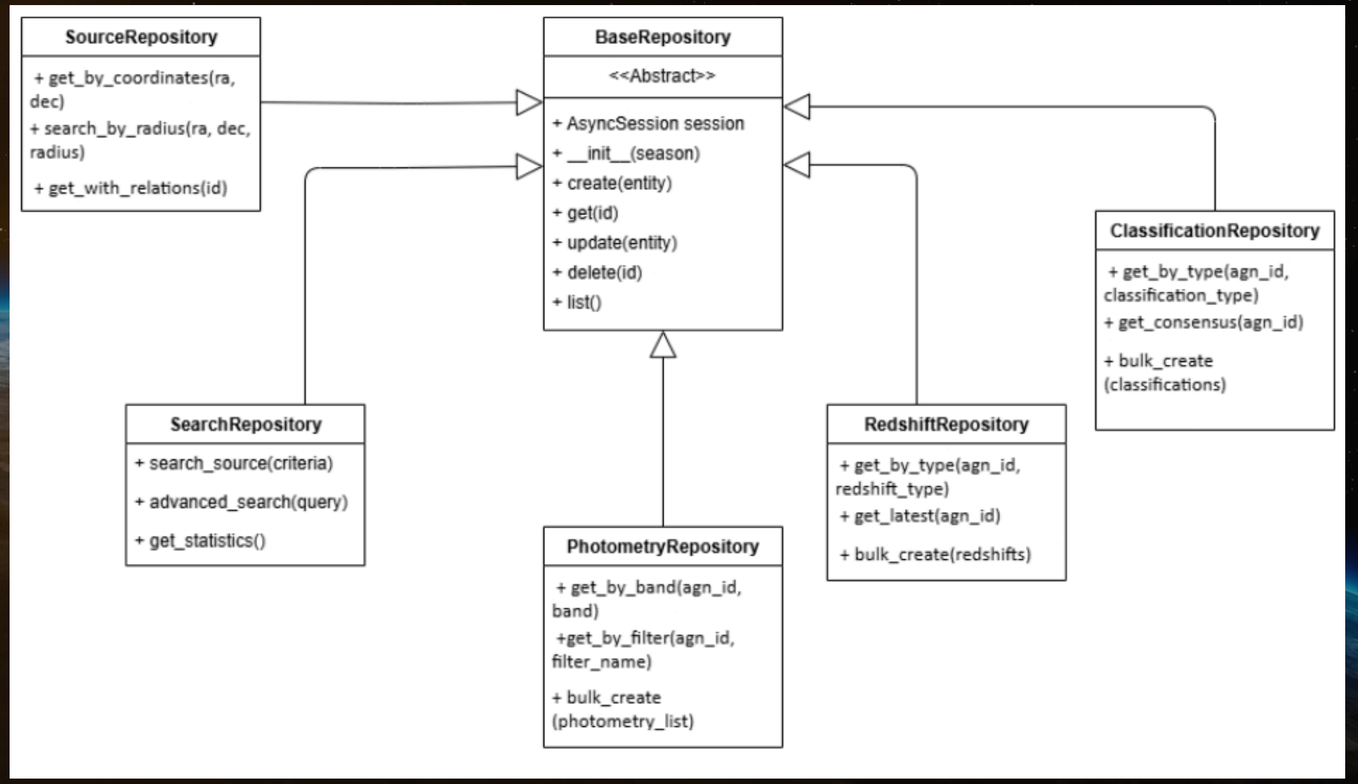
Domain-Specific Repositories

- Extend the base and add specialized queries:
- Examples:
  - *SourceRepository*: get by coordinates
  - *PhotometryRepository*: filter by band
  - *ClassificationRepository*: get consensus classifications

03

Benefits

- Promotes **code reuse** and **separation of concerns.**
- Enhances **maintainability** and modular growth of data access logic.



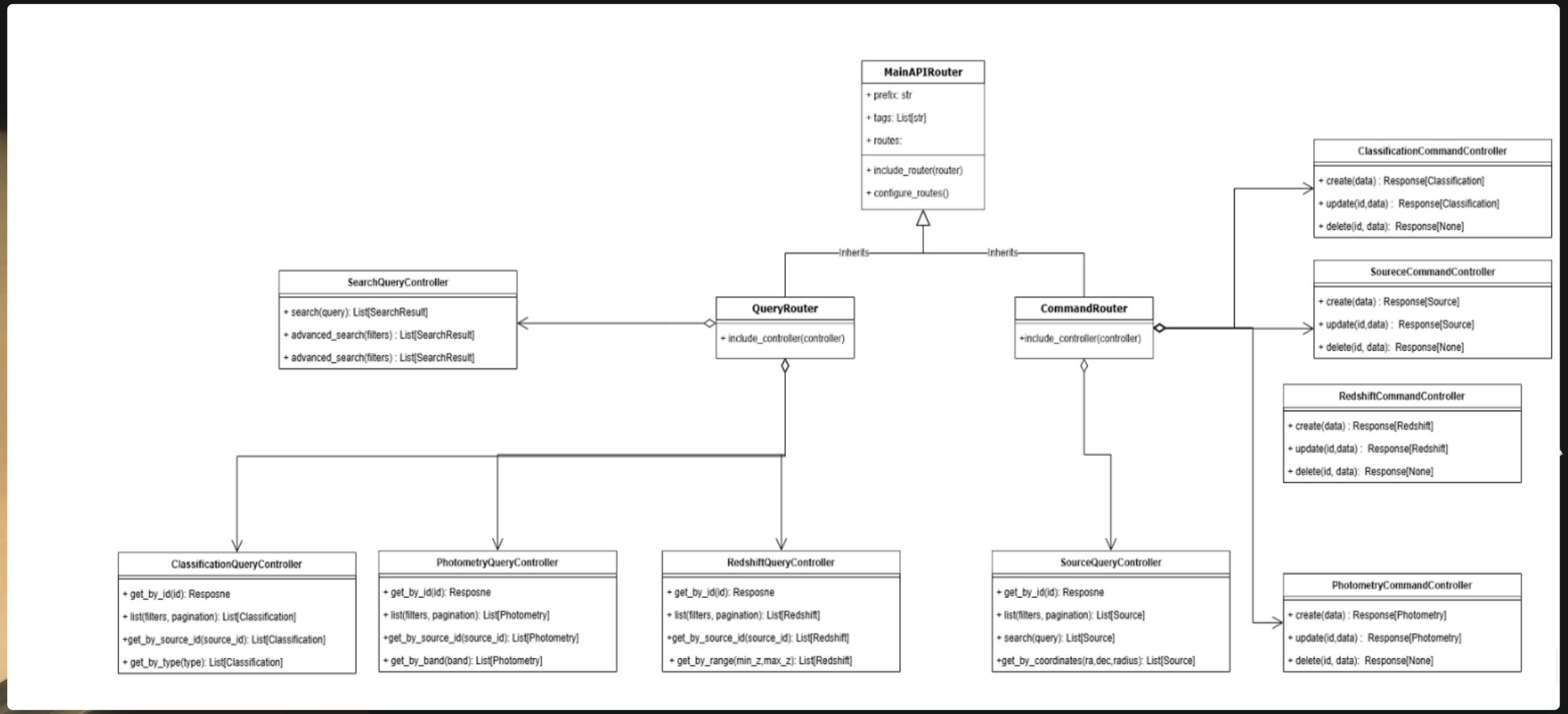
3.3 - API LAYER CLASS DIAGRAM

01 CQRS API Structure

02 Command Controllers

03 Query Controllers

04 Benefits



THANK YOU!