Ontology-Based Website Generation with AGAS: A DSL-Driven Approach

Diana Teixeira & José Almeida & Alberto Simões
University of Minho, Department of Engineering, ALGORITMI

pg53766@alunos.uminho.pt / jj@di.uminho.pt / alberto.simoes@checkmarx.com

What is AGAS?

AGAS (**Application for Automatic Site Generation**) is a platform designed to bridge the gap between structured data representation and web visualization. *Since developing website prototypes is often a time-intensive process, which requires a significant manual effort to apply design, structure and even basic functionalities.*

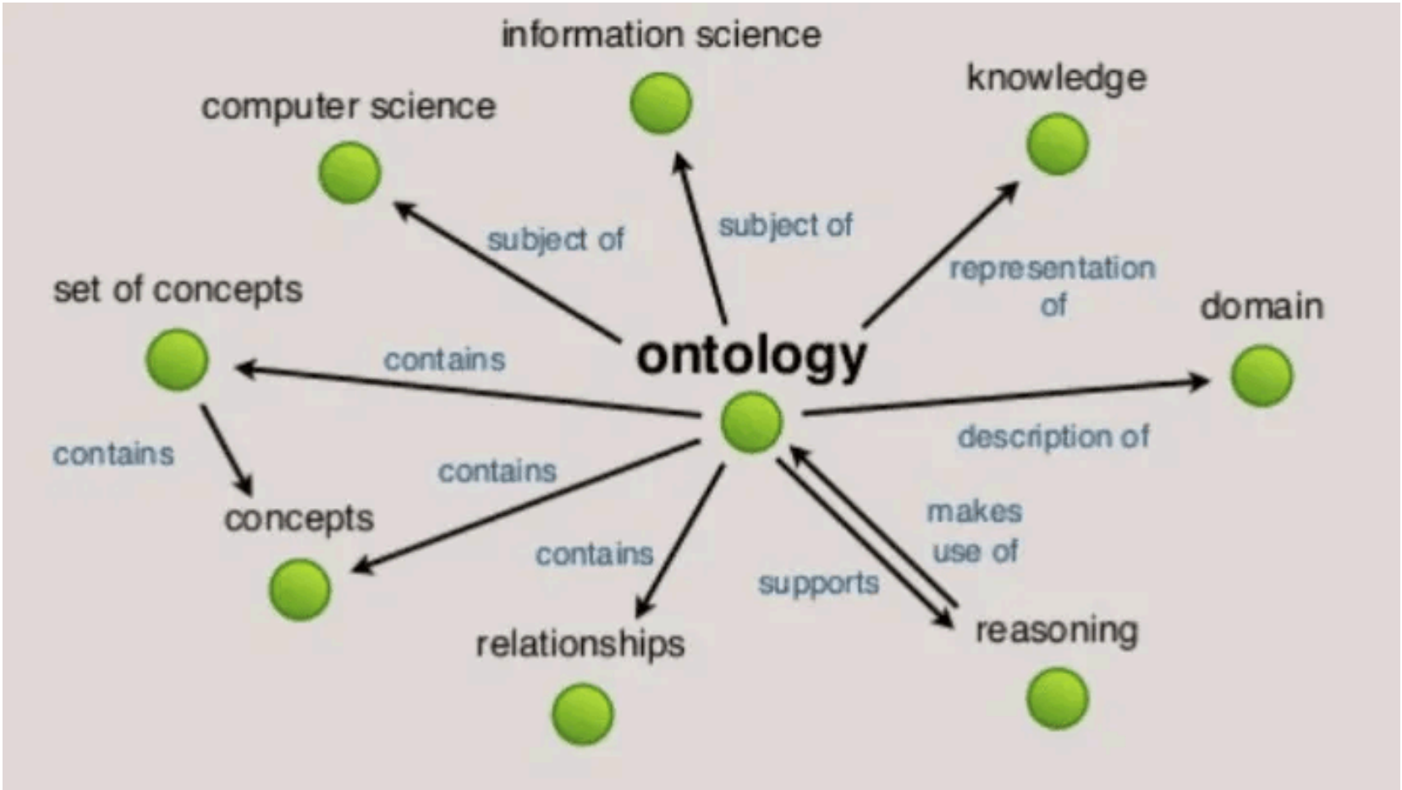
At its core, its goal is to enable:

- Users to define how an **ontology** should be rendered and navigated through a **custom configuration file**;
- The **system** to generate, based on that information, a **web interface**.

This interface will not only display the ontology's information but also make it so it is navigable and interactive, offering users an intuitive way to explore and engage the data.

Why Ontologies?

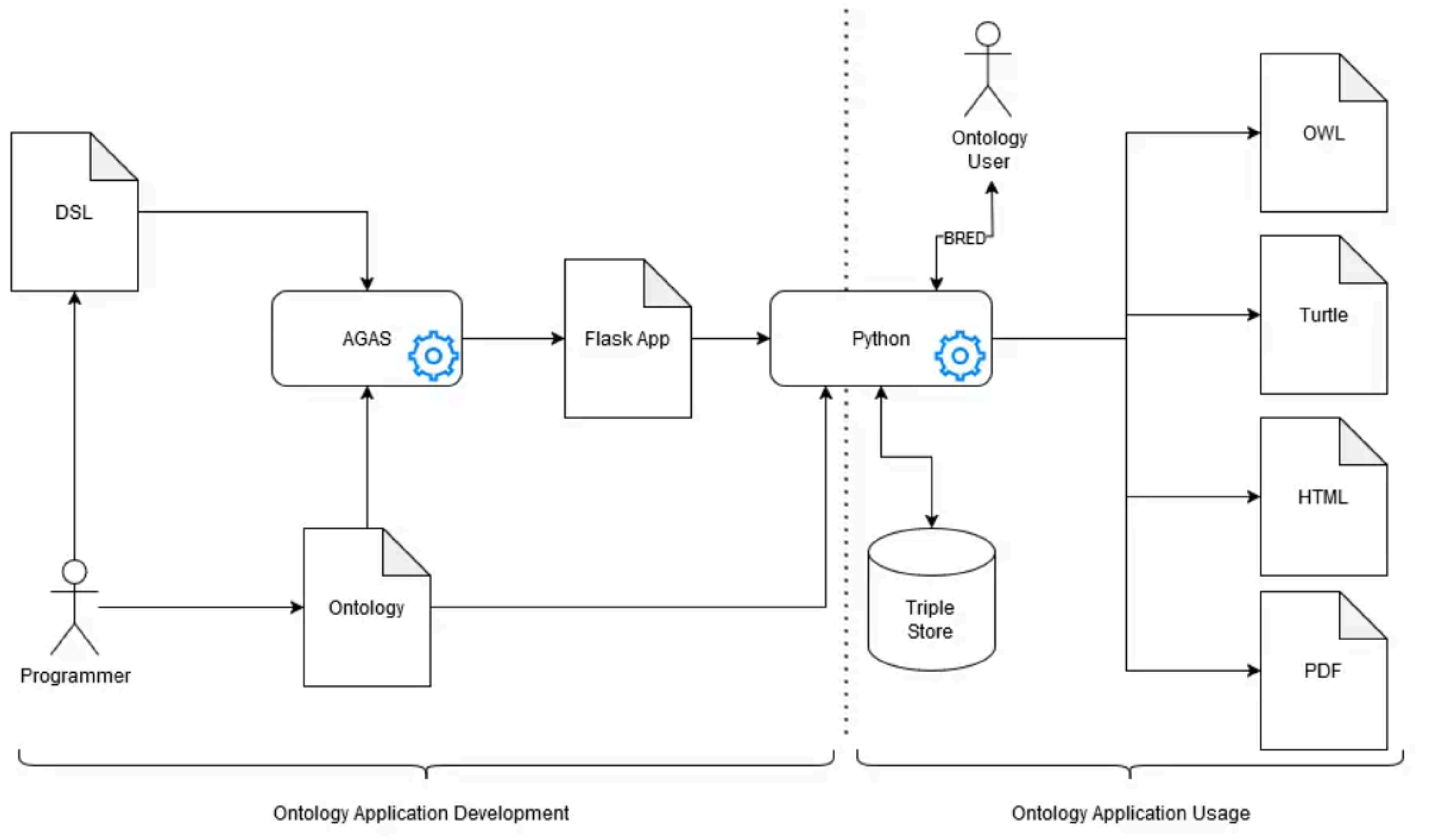
Ontologies of the non-philosophical sense, are known as tools to "**capture knowledge about some domain of interest**" and play a central role in the semantic web by allowing users to represent knowledge in a structured, reusable and meaningful way. *Making them ideal for generating websites where navigation, data, and relations all follow the logic of the domain itself.*



Why is AGAS needed?

The idea for AGAS emerged from a practical need – **building website prototypes quickly based on existing domain knowledge.** *It helps non-programmers turn structured knowledge into usable prototypes quickly. Normally, building something from an ontology requires a lot of coding, but AGAS makes it more accessible.*

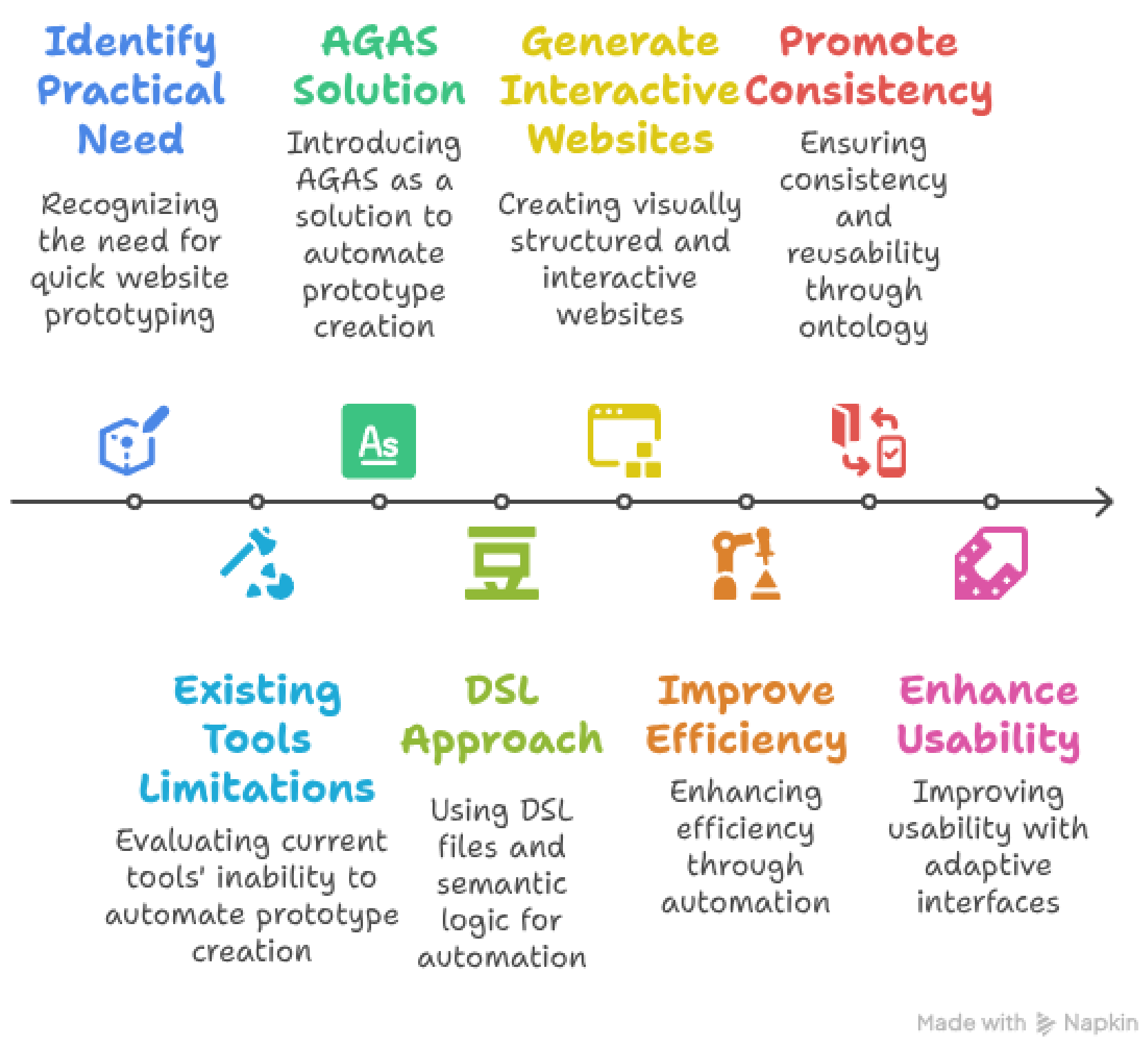
And from the fact that **pre-existing tools**, such as *Figma, Adobe XD, Webflow, Protégé/WebProtégé, Ontospy and GraphDB* are either **too technical or too limited**. *They lack mechanisms to automate the creation of interactive prototypes directly from semantic models. These platforms rarely account for ontology-encoded assets, such as media, images, or custom annotations, resulting in limited expressiveness and usability in real-world applications.*



Why is a DSL needed?

A **Domain-Specific Language (DSL)** is a specialized programming or specification language tailored to address problems within a specific domain that **allows the user specify what they want**.

Then **AGAS**, who is using the DSL as a **bridge between developers and domain experts** to close gaps in **communication**, processes those instructions and **builds a prototype web interface dynamically from the ontology**.



Identify Practical Need
Recognizing the need for quick website prototyping

AGAS Solution
Introducing AGAS as a solution to automate prototype creation

Generate Interactive Websites
Creating visually structured and interactive websites

Promote Consistency
Ensuring consistency and reusability through ontology

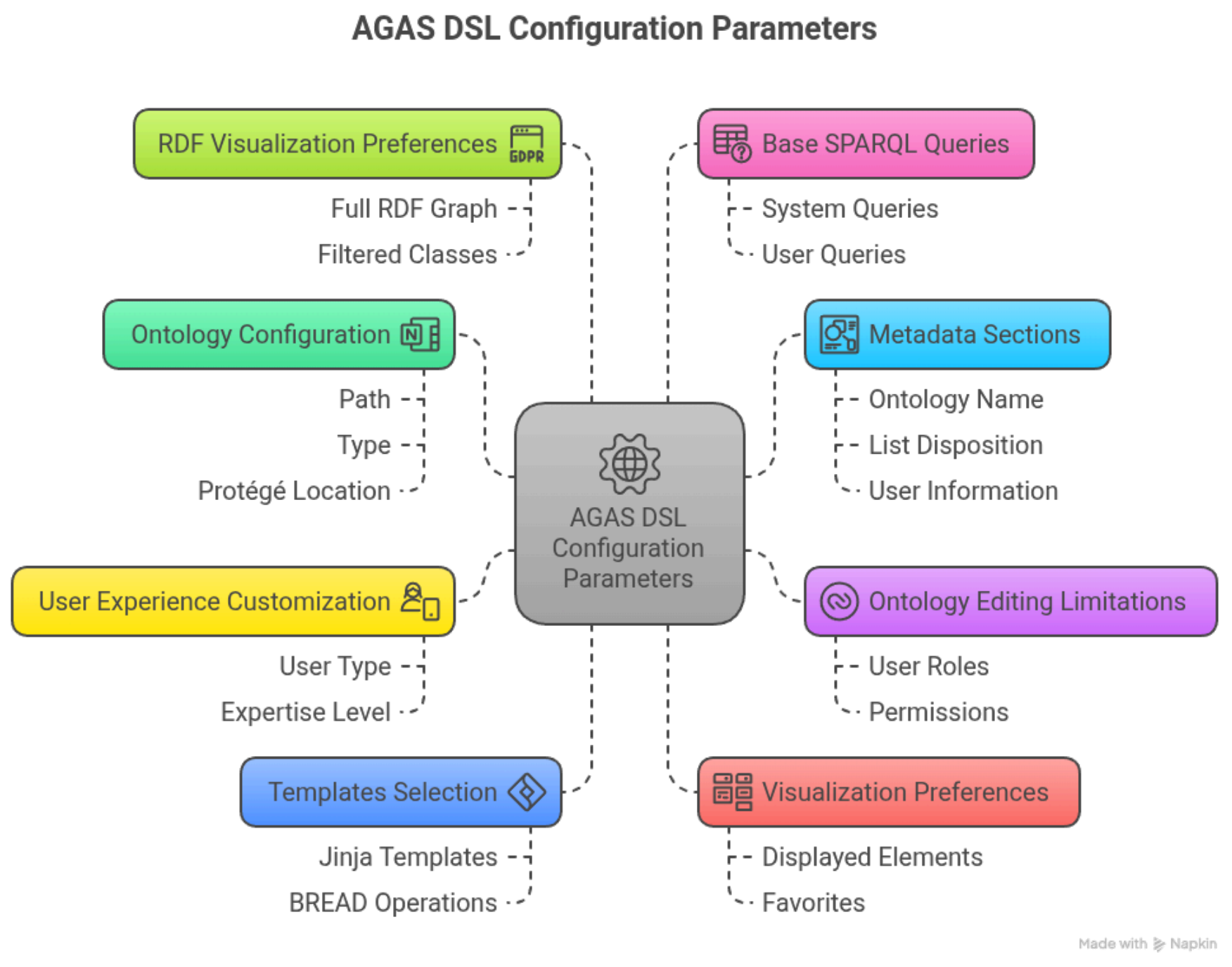
Existing Tools Limitations
Evaluating current tools' inability to automate prototype creation

DSL Approach
Using DSL files and semantic logic for automation

Improve Efficiency
Enhancing efficiency through automation

Enhance Usability
Improving usability with adaptive interfaces

Made with Napkin



AGAS DSL Configuration Parameters

AGAS DSL Configuration Parameters

RDF Visualization Preferences
- Full RDF Graph
- Filtered Classes

Ontology Configuration
- Path
- Type
- Protégé Location

User Experience Customization
- User Type
- Expertise Level

Templates Selection
- Jinja Templates
- BREAD Operations

Base SPARQL Queries
- System Queries
- User Queries

Metadata Sections
- Ontology Name
- List Disposition
- User Information

Ontology Editing Limitations
- User Roles
- Permissions

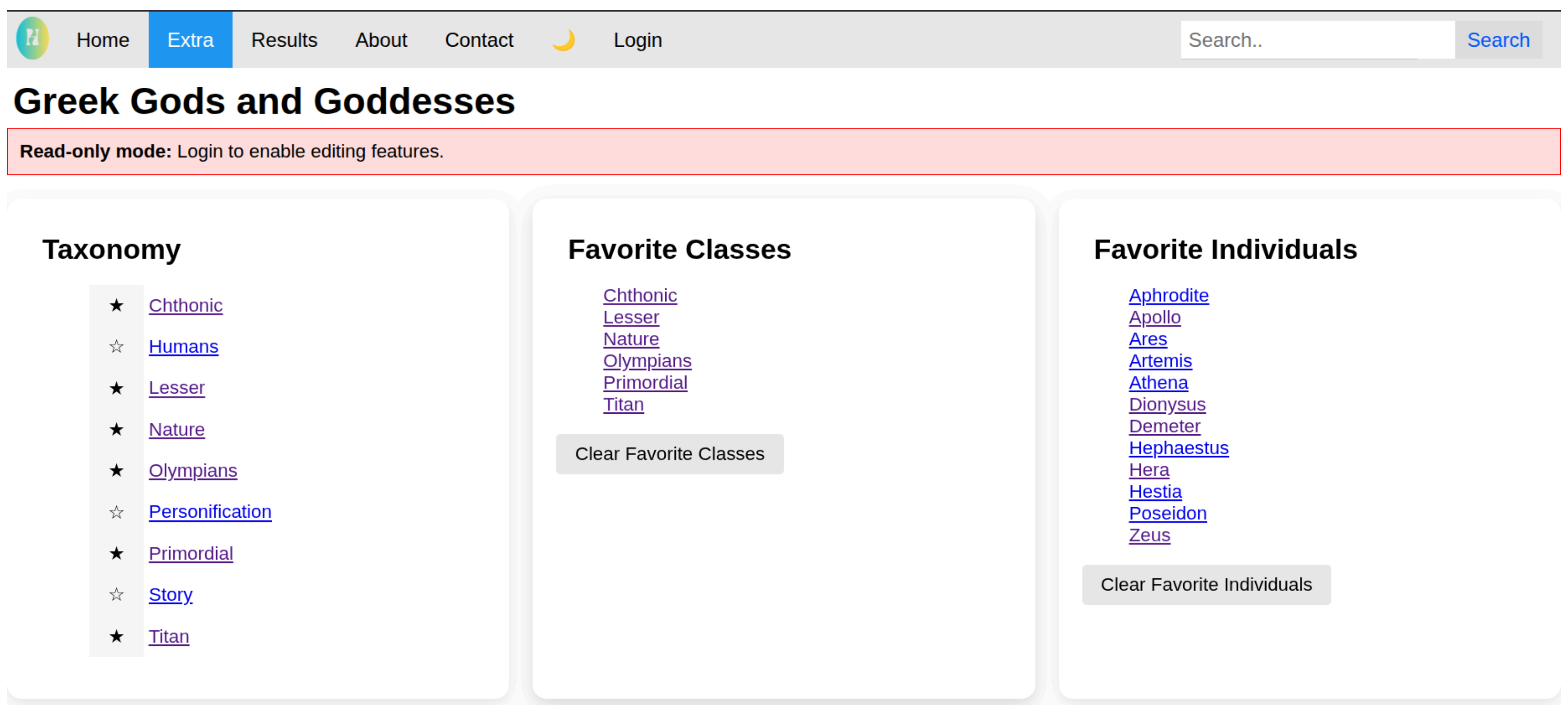
Visualization Preferences
- Displayed Elements
- Favorites

Made with Napkin

What have we done so far?

AGAS is **still under development**, but its core architecture is already defined and implemented across **3** interconnected layers:

1. ***Configuration*** – *More specifically the **DSL**;*
2. ***Processing*** – *Which entails the app in itself, more specifically the **Python** – used so I could integrate everything smoothly in one language instead of stitching multiple environments together, **Lark** – lightweight but powerful parsing library that was simple to set up, and **SPARQL** – standard way to query ontologies;*
3. ***Interface*** – *Which entails the rendering and visualization of the ontology information, following configuration instructions, using **Flask**, which is lightweight and flexible, and consequently perfect for prototyping.*



Greek Gods and Goddesses

Read-only mode: Login to enable editing features.

Taxonomy

- ★ Chthonic
- ☆ Humans
- ★ Lesser
- ★ Nature
- ★ Olympians
- ☆ Personification
- ★ Primordial
- ☆ Story
- ★ Titan

Favorite Classes

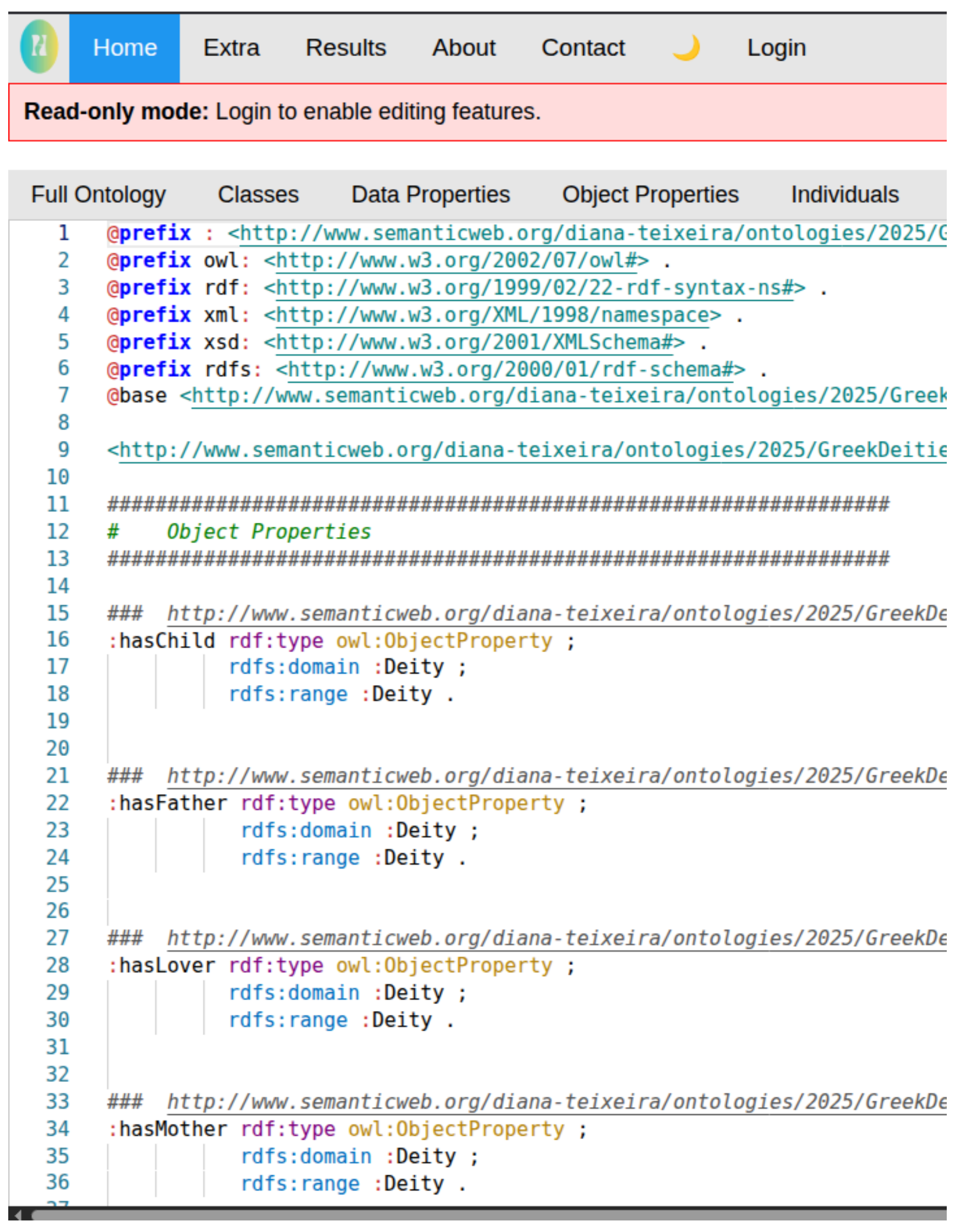
Chthonic
Lesser
Nature
Olympians
Primordial
Titan

Clear Favorite Classes

Favorite Individuals

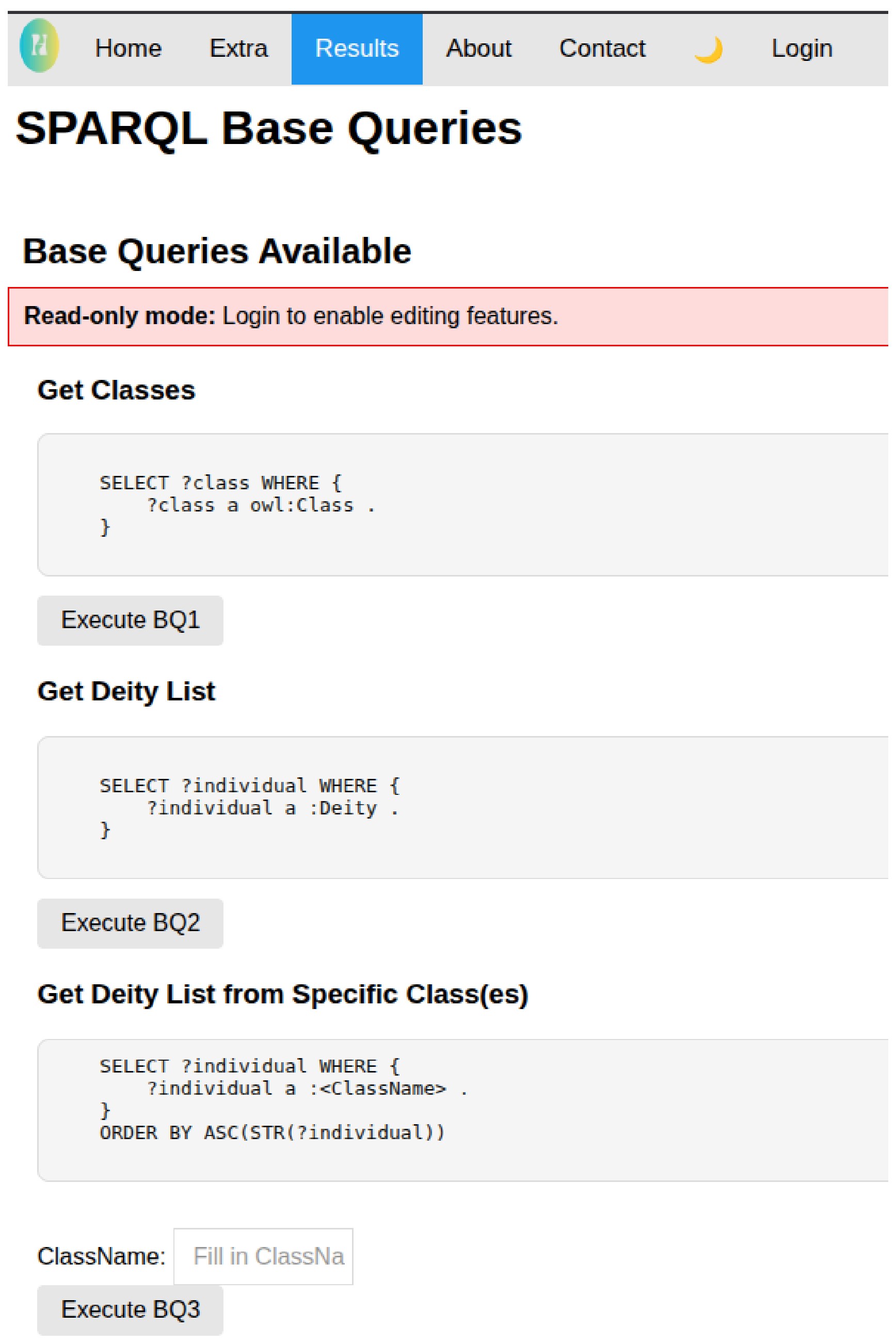
Aphrodite
Apollo
Ares
Artemis
Athena
Dionysus
Demeter
Hephaestus
Hera
Hestia
Poseidon
Zeus

Clear Favorite Individuals



Read-only mode: Login to enable editing features.

```
@prefix : <http://www.semanticweb.org/diana-teixeira/ontologies/2025/G
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.semanticweb.org/diana-teixeira/ontologies/2025/Greek

<http://www.semanticweb.org/diana-teixeira/ontologies/2025/GreekDeitie

#################################################################
#    Object Properties
#################################################################

###  http://www.semanticweb.org/diana-teixeira/ontologies/2025/GreekDe
:hasChild rdf:type owl:ObjectProperty ;
          rdfs:domain :Deity ;
          rdfs:range :Deity .


###  http://www.semanticweb.org/diana-teixeira/ontologies/2025/GreekDe
:hasFather rdf:type owl:ObjectProperty ;
           rdfs:domain :Deity ;
           rdfs:range :Deity .


###  http://www.semanticweb.org/diana-teixeira/ontologies/2025/GreekDe
:hasLover rdf:type owl:ObjectProperty ;
          rdfs:domain :Deity ;
          rdfs:range :Deity .


###  http://www.semanticweb.org/diana-teixeira/ontologies/2025/GreekDe
:hasMother rdf:type owl:ObjectProperty ;
           rdfs:domain :Deity ;
           rdfs:range :Deity .
```



SPARQL Base Queries

Base Queries Available

Read-only mode: Login to enable editing features.

Get Classes

```
SELECT ?class WHERE {
    ?class a owl:Class .
}
```

Execute BQ1

Get Deity List

```
SELECT ?individual WHERE {
    ?individual a :Deity .
}
```

Execute BQ2

Get Deity List from Specific Class(es)

```
SELECT ?individual WHERE {
    ?individual a :<ClassName> .
}
ORDER BY ASC(STR(?individual))
```

ClassName: Fill in ClassNa

Execute BQ3

Our Miracle Website Formula

To achieve all of this and **turn complex ontologies into user-friendly websites**, AGAS relies on a **flexible**, layered system that's **low code** and brings together:

1. **Ontology** – semantic data *(classes, taxonomy, properties, deduced information, navegation, query-language)*
2. **AGAS library** – reusable component library *(default, generic and versitile Templates and Widgets, functions, etc)*
3. **DSL** – configuration file *(configurate and appoint values: Ontology Configurations, Editing Permitions, User Experience, Templates Selection, Visualization Preferences, Graph Visualization Preferences, SPARQL Queries, Metadata)*