The Martian Explorer
Chaos to Cosmos

Annie Bhalla
3638974

A. Project Overview

The Martian Explorer is a semantic information system designed to extract, structure, and explore data from Mars missions sourced from authoritative agencies such as NASA and ESA. The core motivation behind the project lies in bridging the gap between unstructured web-based content—such as HTML mission reports, RSS feeds, and datasets—and structured, queryable knowledge representations for space enthusiasts like me, who love tapping into the world of cosmos. Here is an initiative - ***Chaos to Cosmos***.

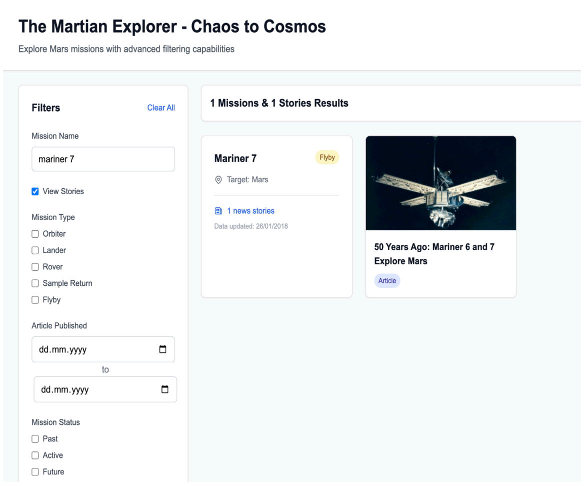
At the core of the system is a data pipeline that transforms heterogeneous content into schema-compliant XML. This structured format is validated using a custom XML Schema Definition (XSD), ensuring semantic consistency and type safety. The validated XML is then persisted in an XML-native database (eXist-db), which offers native support for hierarchical data models and enables expressive querying through XQuery.

The pipeline consists of three modular stages:

1. **Collect** – Scraping mission metadata and reports using Python's requests, BeautifulSoup, and custom feedparser.
2. **Prepare:**
    a. **Transform** – Converting scraped content into structured XML using lxml, mapped precisely to a predefined schema.
    b. **Validate** – Enforcing data correctness by validating against a custom XSD using Python's lxml library. Only schema-compliant files are persisted..
    c. **Store** – Uploading validated XML files into eXist-db using RESTful APIs.
3. **Access** – Querying the stored content via XQuery and rendering filtered mission lists in a React Vite + TailwindCSS frontend.

The frontend supports free-text search, multi-boolean filters, range pickers, and checkbox toggles—allowing users to interactively explore the mission database. For example, the following XQuery snippet retrieves all missions marked as past:

```
for $mission in //mission
     where string($mission/missions_status) = "past"
return $mission
```



The Martian Explorer - Chaos to Cosmos

Explore Mars missions with advanced filtering capabilities

Filters — Clear All

Mission Name: mariner 7

[✓ View Stories]

Mission Type
- Orbiter
- Lander
- Rover
- Sample Return
- Flyby

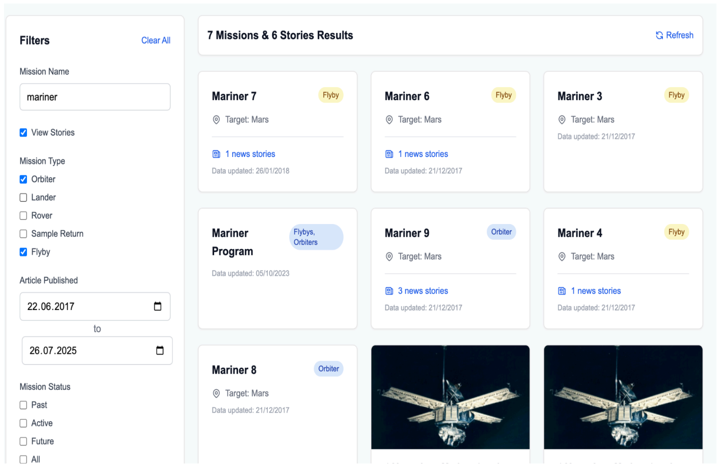
Article Published: dd.mm.yyyy to dd.mm.yyyy

Mission Status
- Past
- Active
- Future

1 Missions & 1 Stories Results

Mariner 7 — Flyby
Target: Mars
1 news stories
Data updated: 26/01/2018

50 Years Ago: Mariner 6 and 7 Explore Mars — Article



Filters — Clear All

Mission Name: mariner

[✓ View Stories]

Mission Type
- [✓ Orbiter]
- Lander
- Rover
- Sample Return
- [✓ Flyby]

Article Published: 22.06.2017 to 26.07.2025

Mission Status
- Past
- Active
- Future
- All

7 Missions & 6 Stories Results — Refresh

Mariner 7 — Flyby
Target: Mars
1 news stories
Data updated: 26/01/2018

Mariner 6 — Flyby
Target: Mars
1 news stories
Data updated: 21/12/2017

Mariner 3 — Flyby
Target: Mars
Data updated: 21/12/2017

Mariner Program — Flybys, Orbiters
Data updated: 05/10/2023

Mariner 9 — Orbiter
Target: Mars
3 news stories
Data updated: 21/12/2017

Mariner 4 — Flyby
Target: Mars
1 news stories
Data updated: 21/12/2017

Mariner 8 — Orbiter
Target: Mars
Data updated: 21/12/2017

Fig: Snippet of UI showing multi-boolean filter for structured exploration

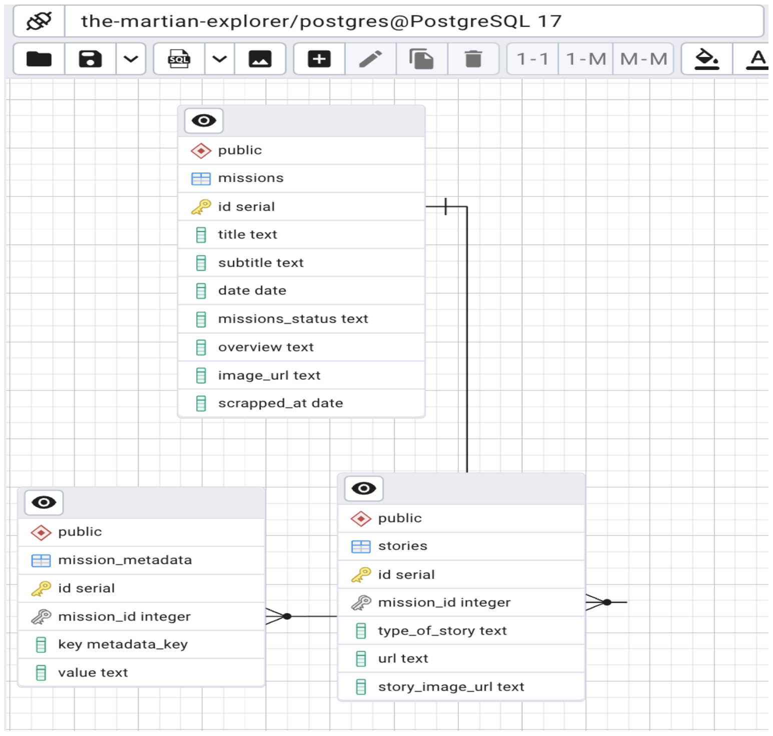
B. RDBMS vs. XML-native database

Q. What would change if a Relational Database were used instead of XMML Database?

- In this project, an XML-native database (eXist-db) was chosen due to the semi-structured nature of Mars mission data, which often varies in depth, nesting, and availability of fields across sources. During implementation, I found that adapting mission data into XML aligned more naturally with the irregularities in NASA's reports, where fields were often optional, inconsistently ordered, or deeply nested. Attempting to predefine strict SQL tables upfront would have limited the system's adaptability and increased overhead in handling edge cases. XML, on the other hand, provided a natural fit for capturing this heterogeneity, enabling hierarchical modeling and schema validation via XSD.
  However, if the system were redesigned using a relational database (RDBMS) such as PostgreSQL or MySQL, significant architectural and conceptual shifts would be necessary in the data model, transformation pipeline, and query interface.

<u>Schema Design in a Relational Database</u>

In a Relational database, data needs to be normalized into flat tables with fixed schemas. Here is a proposed relational schema for storing Mars mission metadata:



the-martian-explorer/postgres@PostgreSQL 17

public
missions
- id serial
- title text
- subtitle text
- date date
- missions_status text
- overview text
- image_url text
- scrapped_at date

public
mission_metadata
- id serial
- mission_id integer
- key metadata_key
- value text

public
stories
- id serial
- mission_id integer
- type_of_story text
- url text
- story_image_url text

Relationships in this model are maintained through foreign keys (e.g., mission_id), allowing for efficient joins between missions and their associated metadata or stories. This normalized structure works well for our current scope, which focuses solely on Mars missions. However, if I were to expand the database to include other mission targets—such as planets, asteroids, comets, or other celestial bodies—the relational schema would need to be rethought and redesigned to accommodate additional entities and relationships.

Changes in Prepare Phase

- The XML transformation and XSD validation steps would be removed in favor of schema enforcement using RDBMS constraints (e.g., NOT NULL, UNIQUE, FOREIGN KEY, CHECK).
- Data ingestion scripts would transform scraped content into SQL INSERT statements instead of XML document upload to database.
- Handling nested or optional fields—such as instruments or mission objectives—would require flattening them into dedicated tables or representing them using flexible key-value pairs, which complicates the schema.

Example Query in SQL

Lets see a query to find all rover missions after 2018 with methane analysis capabilities. This is a type of multi boolean filter:

PostgreSQL	XQuery

SELECT m.title, m.date
FROM Missions m
JOIN Metadata md ON m.id = md.mission_id
WHERE md.key = 'Type' AND md.value ILIKE '%rover%'
  AND m.date >= '2019-01-01'
  AND EXISTS (
    SELECT 1 FROM Metadata md2
    WHERE md2.mission_id = m.id
      AND md2.key = 'Objective'
      AND md2.value ILIKE '%methane%'
  );
	for $mission in //mission
	where
	
	contains(lower-case($mission/metadata_table/metadata[key='Type']/value), 'rover')
	  and
	xs:date(substring-before(string($mission/date), "T")) >= xs:date("2019-01-01")
	  and some $m in
	$mission/metadata_table/metadata
	      satisfies (
	        $m/key = 'Objective'
	        and contains(lower-case($m/value), 'methane')
	      )
	return $mission

SQL query uses SQL joins and subqueries to achieve what XQuery (shown in the table on right) would handle natively via hierarchical XML.

Implications of the Change: RDBMS

Pros:

a. Mature tooling and performance optimizations
b. Easier to integrate with business intelligence tools
c. SQL familiarity among developers

Cons:

a. Poor fit for highly nested, variable, or document-centric data
b. Schema evolution is costly; adding new fields or relationships requires migrations
c. Requires reconstruction of hierarchies through joins, impacting query readability and performance

**** Hybrid Architecture Potential ****
In future iterations, a hybrid architecture could be explored. For instance, PostgreSQL could be used to store flat metadata for analytical purposes (e.g., mission statistics, filtering), while eXist-db could retain full XML documents and nested reports for semantic exploration and full-text search. This would offer a balanced trade-off between structure, performance, and flexibility.

Conclusion

While a relational model is viable, it imposes structural constraints that conflict with the real-world complexity and variability of scientific mission data. The current XML + eXist-db solution aligns better with the data's hierarchical nature and the need for flexible, schema-valid exploration.
For long-term scalability, especially where analytics and uniform data aggregation are involved, a dual-model approach could offer the best of both paradigms.

C. References

1. https://www.w3schools.com/xml/xml_whatis.asp
2. https://www.w3schools.com/xml/xquery_intro.asp
3. https://www.w3schools.com/xml/xpath_intro.asp
4. https://www.postgresql.org/
5. https://react.dev/
6. https://www.typescriptlang.org/
7. https://www.docker.com/
8. https://science.nasa.gov/science-missions/
9. Github Reference: https://github.com/Anniebhalla16/TheMartianExplorer