MAINTAINING MICROSERVICES: APPLICATIONS FOR THE ASTROMATERIALS DATA SYSTEM

Planetary Science Informatics and Data Analytics (PSIDA) conference

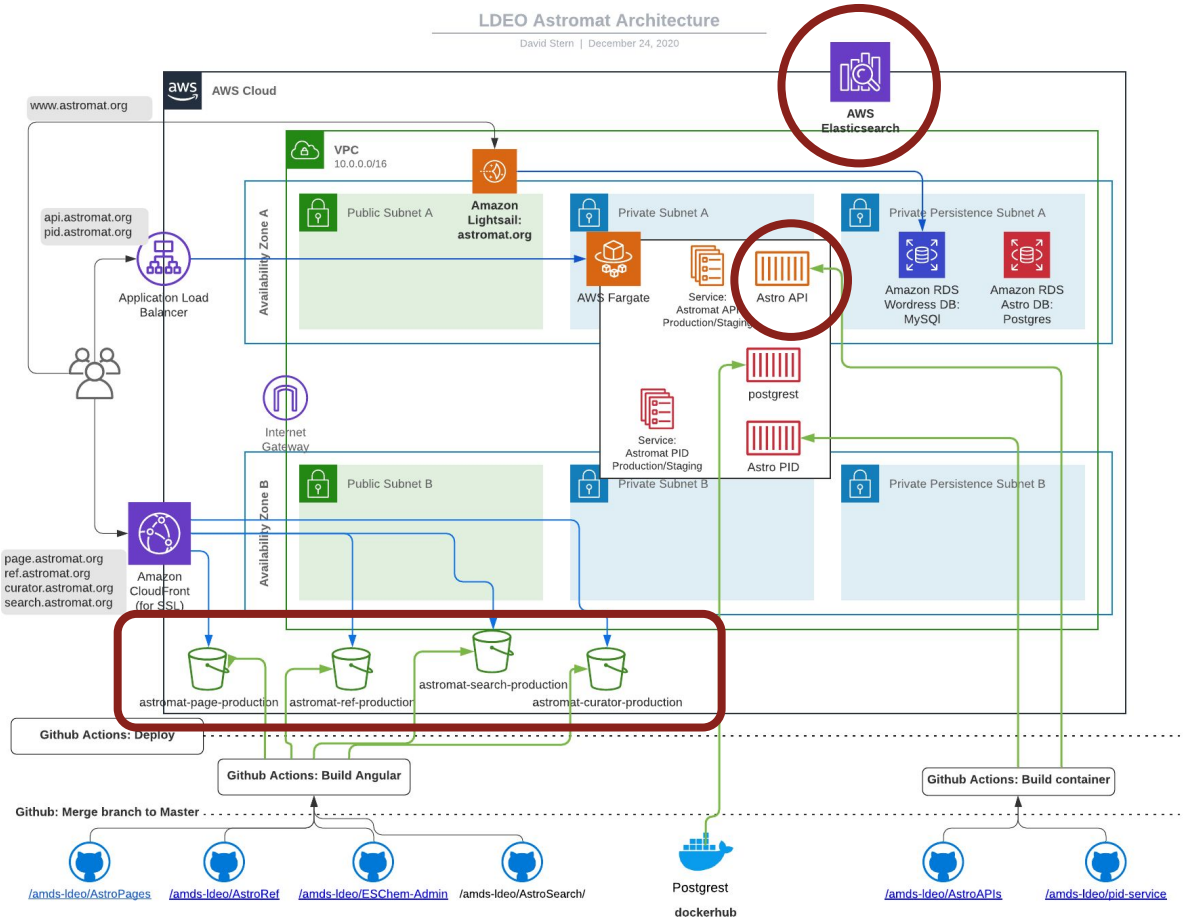
June 23rd, 2022

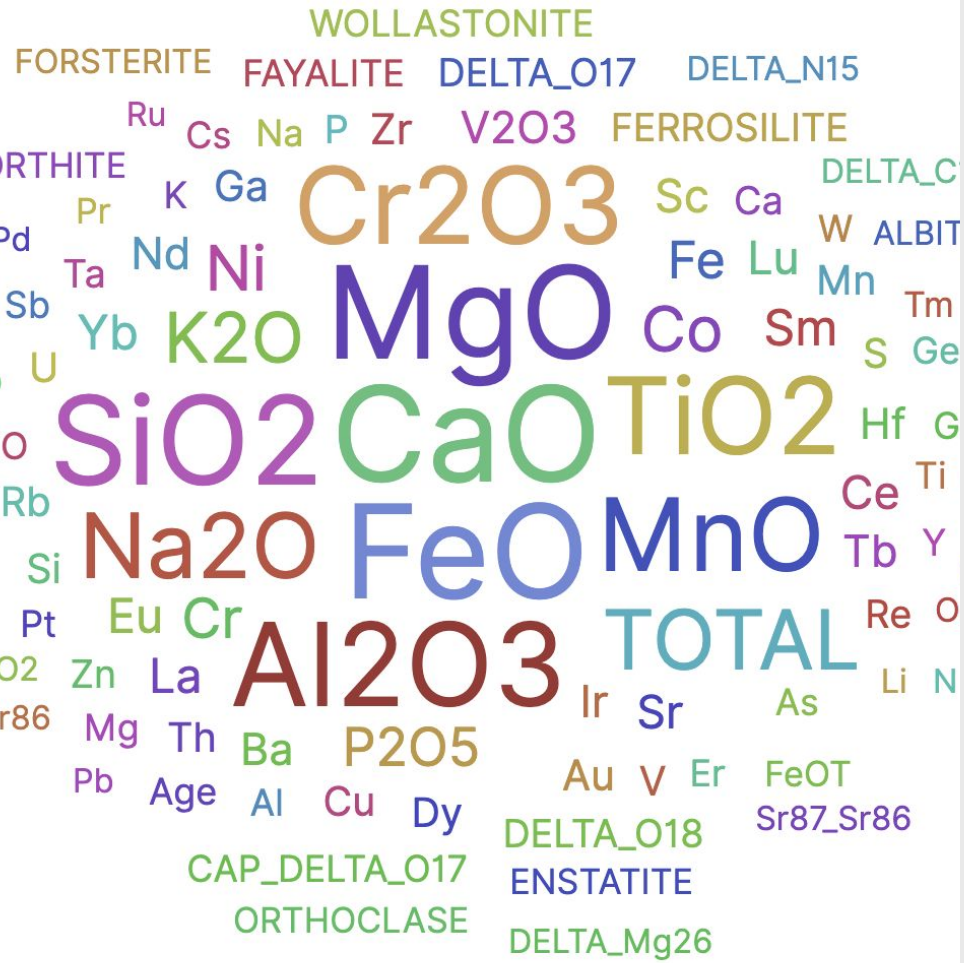
J.D. Figueroa, K.Lehnert, P. Ji, L. Profeta, J. Mays, A. Johansson,

Lamont-Doherty Earth Observatory, Columbia University, New York, USA

What is AstroMat?

Astromaterial Data System (AstroMat) is a comprehensive data system for laboratory analytical data generated by the study of astromaterials. It is designed as an ecosystem of interconnected applications that provide human- and machine-readable interfaces to the data gathered and managed in AstroMat's databases.

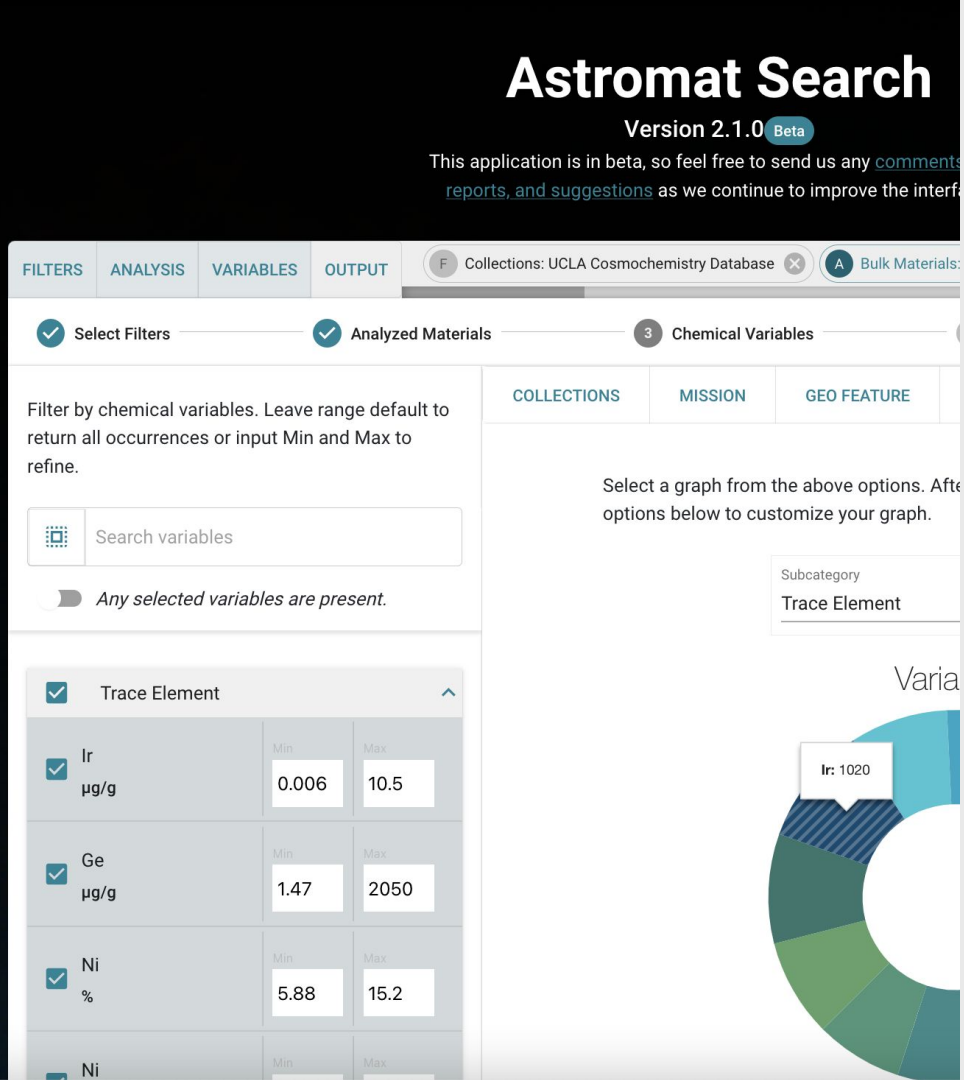




AstroMat Data

With 777,896 chemical data points for lunar (349,610) and meteorite (394,452) collections consisting of 51% mineral analysis, 42.74% rock analysis and 6.2% inclusion analysis, the AstroMat data system aims to become a one stop-shop for curated astromaterials data.

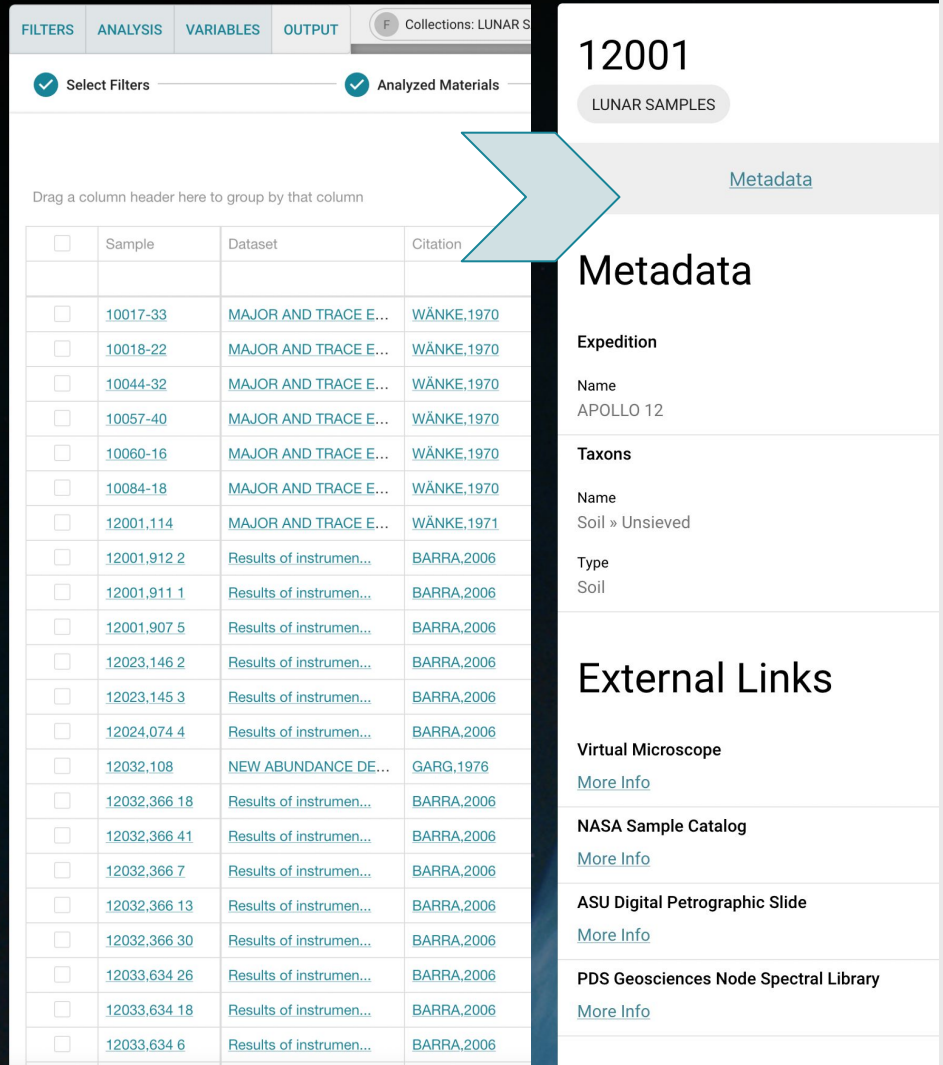
— Text cloud of chemical variables as they exist in AstroMat.



AstroMat Faceted Search

Pre-built static javascript applications using functional programming techniques deliver modular code used to deliver user-friendly front-end applications with quickly accessible data.

The flow of data comes from relational databases, to an elasticsearch index, to API, to front-end applications, and finally to end user(s).



Landing Page Philosophy

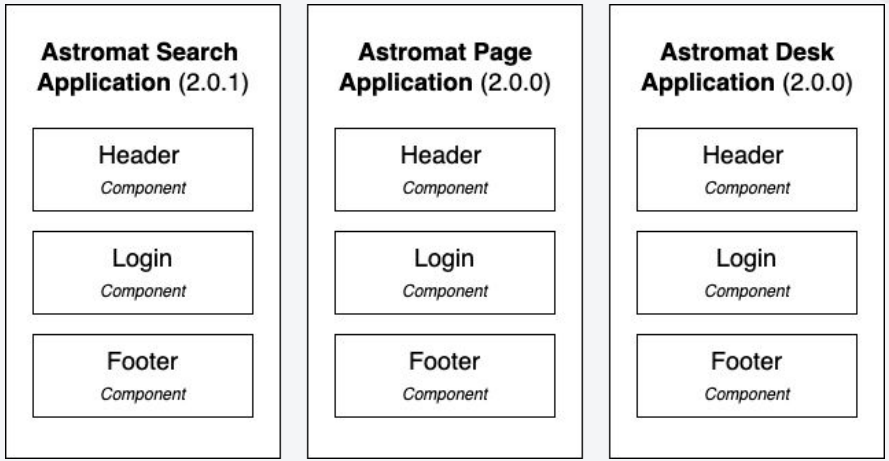
After search and export features, AstroMat looks forward to designing dynamic parent sample pages that contain aggregate metadata and links to appropriate external resources.

Our landing pages will also include a hierarchical representation of all samples and aggregate sample data representing analysis done and sample sub-components.

Maintaining Front-End Applications

Employing the Modular Federation Approach

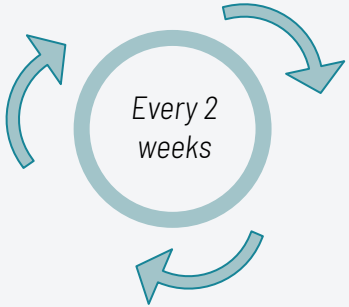
Current State of Frontend Applications



- Request data from API resource, which pulls data from an elasticsearch index.
- Single Page Applications (SPAs) all have duplicate components including header, navigation, login, and footer components.
- All SPAs are stored in AWS S3 buckets through continuous deployment.
- All applications require similar design, usability, and interoperability.

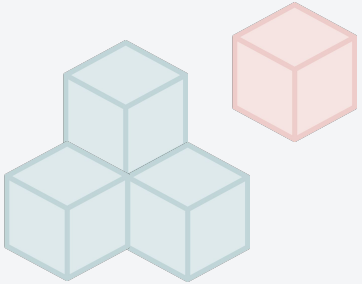
Why Modular Applications?

Agile
Sprint Cycle



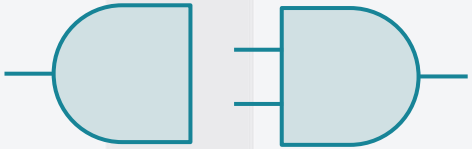
Compartmentalization of project management

Atomic
Design Methodology



Design theory aims to treat applications as component pieces.

API - Driven
GET, PUT, POST, DELETE



APIs distribute data according to application endpoint/query specifications

The Benefits of Modular Architectures

- Simple to update and refactor application components.
- Decrease complexity makes it easier to maintain.
- Enhanced collaboration since developers can focus on pieces rather than the entire system.
- Readable code due to decreased complexity and functional programming approach.
- Bugs are easier to detect and address.
- Modular components are reusable.
- Iterative development and continuous deployment.

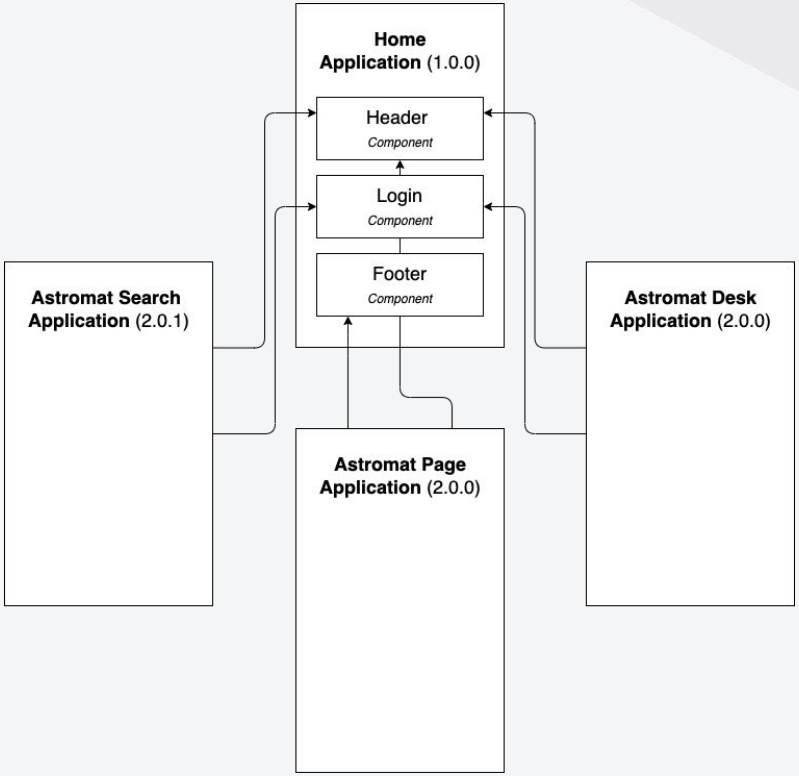
Moving towards ***distributed***, ***decentralized***, ***dynamic***, and ***cloud-based*** web applications.

The Drawbacks of Modular Architectures

- Higher initial investment.
- Greater planning complexity.
- Continuous monitoring to maintain codebase.

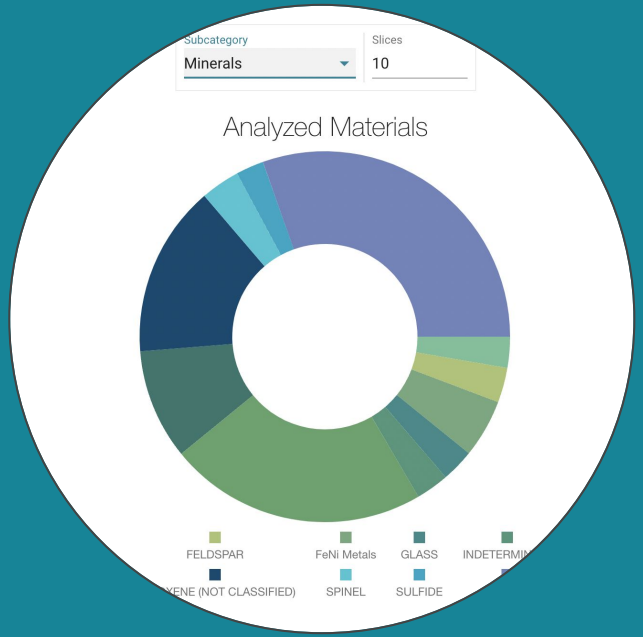
Ok, so modular is good. Now what?

Modular Federation - Home Application



- All applications are still independent of one another, but gather components from the *Home Application* which hosts the shared components (Header, Footer, Login, etc.)
- In this scenario, modular federation is taken from a built and deployed static asset store (AWS S3).
- Home App modular components are shared through a remote entry webpack configuration.

Multiple components are taken from a single asset store to feed modular applications.



In conclusion...

Micro-frontends Enhance Maintenance of Modular Applications

Although caveats exist, such as initial investment and planning requirements, designing micro-frontends to be consumed by modular applications can effectively reduce the maintenance costs of updating reusable components while maintaining cohesive and non-duplicative code. Projects looking to maintain API-driven single page applications should consider implementing modular federation for ease and efficiency of maintenance.

jdf@ldeo.columbia.edu