GPP Technology Choices
Replaces a Quarter Century of Cruft
Service Architecture

- GPP is structured as a collection of client web apps communicating with cloud-deployed services using GraphQL queries.
- Allows services to evolve independently and simplifies upgrades.
- Each service may utilize a database to support its persistence requirements.
- Services may use other GPP services or even external services to fulfill requests.
Core Programming Language and Development Paradigm
Functional Programming

● Building software by composing and applying pure functions, avoiding shared mutable state.
● A declarative style of programming where expressions are employed rather than imperative statements.
● Why
  ○ Pure functions are easier to reason about and to test.
  ○ Encourages reuse yielding reliable, modular systems.
● Alternatives
  ○ Traditional imperative programming (for example in Java).
  ○ In our experience building large systems on top of shared mutable state leads to serious maintenance issues.
Scala

- A statically typed, functional programming language on the JVM.
- Why
  - Static typing helps avoid bugs and simplifies long-term maintenance.
  - Builds on the Gemini software team’s years of experience on the JVM.
  - Robust ecosystem with ready-made solutions to many common issues.
- Alternatives
  - Eta and Frege are other statically typed functional program languages that target the JVM but both are more similar to Haskell, which is less familiar to the high-level group.
  - Neither has the level of adoption that Scala enjoys.
  - Kotlin + Arrow might be an alternative, but it appears to be less mature and as a team we have invested many years in Scala.
Typelevel Ecosystem

- Suite of modular, pure functional Scala libraries that work together including

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cats</td>
<td>Abstractions for FP in Scala</td>
</tr>
<tr>
<td>Cats-Effect</td>
<td>IO Monad and effect handling in general</td>
</tr>
<tr>
<td>Doobie</td>
<td>Functional database access</td>
</tr>
<tr>
<td>FS2</td>
<td>Stream Processing</td>
</tr>
<tr>
<td>Http4s</td>
<td>Functional HTTP client and server library</td>
</tr>
<tr>
<td>Monocle</td>
<td>Simplifies reading and updating hierarchical, immutable structures</td>
</tr>
</tbody>
</table>
Typelevel Ecosystem

- **Why**
  - Scala directly supports functional programming at a superficial level.
  - A foundational library providing abstractions is required (e.g., Functor, Applicative, Monad typeclass definitions, syntax, etc.)
  - The Typelevel Cats library provides this foundation.
  - Having selected a foundation, using functional building blocks that assume the same foundation makes it easier to piece together code.

- **Alternatives**
  - Scalaz is the only real “competitor” to Cats but the industry has mostly adopted the Typelevel ecosystem.
Front End
Scala.js

- Scala to JavaScript compilation.
- Why
  - Extends Scala to the browser, enabling shared library code across client and server.
  - Static typing in the frontend, simplifying maintenance.
  - Provides interoperability with JavaScript libraries.
  - Freedom to move logic from the backend to the frontend when appropriate.
- Alternatives
  - Clients directly written in JavaScript imply all the computation be relegated to the server, or code must be ported and duplicated across both.
  - JavaScript suffers from maintenance issues common to dynamically typed languages in general.
  - Languages like TypeScript/PureScript would solve maintenance issues of JavaScript, but would also require duplicated code.
### Client Side Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>React / scalajs-react</td>
<td>Front end logic structure</td>
</tr>
<tr>
<td>Semantic UI</td>
<td>Visual styling</td>
</tr>
<tr>
<td>Ag-grid</td>
<td>Advanced tables</td>
</tr>
<tr>
<td>Aladin</td>
<td>Catalog visualization</td>
</tr>
<tr>
<td>svg components</td>
<td>Instrument visualization</td>
</tr>
</tbody>
</table>
Backend
**PostgreSQL**

- Free and open source relational database.
- **Why**
  - Reliable, robust, efficient.
  - Excellent community support.
  - Available as a Heroku service.
- **Alternatives**
  - Paid alternatives exist but offer no apparent advantages for our needs.
  - Open-source alternatives like MariaDB are available, and would likely suffice, but Postgres is more familiar and Heroku (covered later) offers a managed Postgres service.
  - NoSQL options are not compatible with our emphasis on data integrity and structure.
APIs (Communications)
GraphQL

- API query language used by client web apps and between services themselves.
- Also offered for one-off advanced-user queries and arbitrary scripts.
- Why
  - Clients can specify exactly the information they require.
  - Simplifies API evolution.
  - Supports subscriptions which allow clients to update upon remote changes.
  - Users may write service queries in any language.
- Alternatives
  - Traditional RESTful APIs are an alternative and may be used in some cases.
  - Client has no control over the result and often receives too much data or not enough (requiring additional queries).
Deployment
Heroku

• A continuous and cloud-based deployment service.

• Why
  ○ Simplifies cloud-platform management over using raw Amazon Web Services.
  ○ Automates application updates.
  ○ Provides multiple environments for testing, staging, and production.
  ○ On-demand resizing and scaling with integrated load-balancing.
  ○ Tight integration with GitHub for testing and deployment

• Alternatives
  ○ Amazon Web Services/Google Cloud/Azure offer similar services but expose all the complexity of managing servers and infrastructure.
  ○ Gemini/NOIRLab does not currently offer cloud based hosting and no managed services over raw VMs.
  ○ These alternatives become considerably more expensive considering management costs.