API Architecture
for the Data Interoperability at OSU initiative

Introduction — Principles and Standards

OSU’s current approach to data interoperability consists of low level access and custom data models for each application. We envision a new architecture that provides loosely coupled interoperability for a variety of data consumers. The new architecture will provide consistent data models, robust documentation, improved security, and near real time integration.

This architecture will give OSU more agility to develop, evolve, and replace services. Departments won't be vendor-locked from the perspective of data integration into a single solution or ecosystem of applications. The loosely coupled, interface-based integrations between systems will mean that services will be swappable.

This architecture will use proven standards, design patterns, and data formats such as HTTPS, REST and JSON over enforcing proprietary, or complex standards like Simple Object Access Protocol or Enterprise Service Bus (ESB). This architecture and Representational State Transfer (REST) interfaces will be initially developed as a project by IS developers within OSU.

As we develop this architecture, we will adhere to these principles:

- We want to fail fast through continuous delivery during development, which will speed the feedback cycle. The implementation project will keep the architecture and restful Application Programming Interfaces (APIs) that are successful and throw away those that aren’t needed or have failed to meet our users’ needs.
- The APIs have the goal to make it easy to do the right thing, but hard or impossible to do the wrong thing. This principle will be accomplished by simplifying the correct choices and leading developers through the best path for using the APIs.
- We believe in open development and that it outweighs the benefits of closed development and security through obscurity.
- The architecture will focus on delivering fewer, but more commonly used API endpoints. The narrow focus will provide better API endpoints rather than trying to solve every business case at the cost of not delivering reliable services.

It is the hope that frequent successful releases will earn and maintain trust with the developer community at OSU. This architecture needs the trust of our end users and we’ll earn it by blogging openly about our successes and failures, reaching out to developers for feedback, and providing resilient services.
API Management
This new API architecture won't discourage OSU developers from writing code. On the contrary, this architecture will allow developers to focus on writing code specifically in the areas that add the most value to the OSU community's business needs.

Ideally, developers will focus on writing APIs instead of writing code for API management, such as: logging, authorization, authentication, metrics, etc. This is why a 3rd party non-OSU developed solution that already excels at API management will be used. Appendix A includes a diagram of the API architecture with API consumers / clients on one end and the API management layer filtering out information, authentication / authorization among other features. The API management will provide logging, analytics, authentication, security policies, throttling, multi-environment (production, development, staging) support, and a developer portal where APIs are documented and presented.
Resiliency

Reliable and resilient services are a high priority for this architecture. Systems and application developers will use the APIs on a daily basis to integrate OSU systems.

In the old integration pattern, a local unencrypted cache of data needed by applications was spread throughout the systems. This local cache provided a mechanism in case the systems that stored the data were not available. The new architecture will use a centralized set of APIs and this layer can provide default responses, use a cache or try a different data source when the primary data source is not available. A reliable and resilient set of APIs will discourage the previous workarounds of caching the data locally.

Figure 1 depicts the data flow through the API architecture. The API Gateway (an appliance or service described in more detailed later) will act as a central hub where the various clients can fetch the information from a variety of services. The clients can vary in type such as: mobile devices, servers, researchers or web applications. The services will be provided either by OSU developers or 3rd party software. This architecture will allow us to decouple the clients and services as well as focus our resiliency efforts on the API gateway and services.

Figure 1 - Data Flow
When to write APIs?
A website will be setup to allow OSU developers to submit ideas and requests for APIs. An advisory group made up of a few developers / representatives on campus will be created to advise on decisions, filter through requests and mitigate issues / disagreements regarding the architecture and community feedback. To keep in line with the continuous delivery goal, the architecture will include APIs for the most commonly requested integrations. However, maintaining and evolving the APIs will require resources. Developing an API for every single IS system is not feasible as a short term goal for the project. The prioritized list of upcoming APIs, will be posted in the developer portal.

When not to write APIs?
If an application already provides a web-based API, we will not write a new one. If the system is going to be retired within one to three years, an API won't be developed for it. One off events / integrations, with a low value proposition and high maintenance costs, won't be provided via web APIs. The advisory group and project team will help with these decisions.

Service Oriented Architecture
SOA is defined by Gartner as “meta-architecture,” an architecture that unifies via services multiple standalone application-level architectures to enable composite applications. The API project will focus on the SOA principles of encapsulation, separation of concerns, and loose coupling. This will be achieved through modularity, as well as distributable, discoverable, swappable and shareable services.

The API architecture will go one step further and strongly recommends a microservice architecture to be used in applications. Microservices are similar to SOA and are an evolution in the architecture design. Microservices is a vague term, but it usually refers to multiple small apps that use HTTPS / REST to communicate, that are meant for a specific purpose and have a small code base. SOA on the other hand is more general in that it specifies that applications should communicate via services that transmit data over the network. SOA could use REST, but it was often implemented via Simple Object Access Protocol, Remote Method Invocation or Remote Procedure Call.

Figure 2 visualizes the topology of a microservice architecture, where apps are made up of multiple microservices. Instead of developing big, monolithic applications, applications should be made up of smaller apps, which interact with each other via services. Each service has a small footprint, is UI neutral, and is focused on a single task. Examples of small microservices would be: bus stops and locations, group list and membership or course list and descriptions. In these microservice examples, the developer creating an interface for bus stops or displaying the list of courses can display and interact with the data in an app running on a phone, a web browser, a backend server or on a wearable device. By providing these interfaces, the API architecture achieves endpoint independence and loosely coupled integrations.
**Data Level Agreements**

Before a system can access the data provided by RESTful APIs, the developer maintaining the application will have to go through a data level agreement process. This approach will allow developers to innovate and develop application integrations without the access to data becoming a bottleneck in the process. The data level agreement will grant the application access to a consistent set of data models (e.g. term, student and course). Systems won't have access or a concept of database tables and columns when integrating with data providers. Systems that have access to a data model may choose to use one or all of the fields within that data model.

This consistent set of data models will allow for concise documentation without the need to document differences between every integration. Integrations that need term or student data will use the respective consistent data models. Developers from different colleges and departments can collaborate with colleagues in regard to their individual integrations because all integrations will use the data level agreements and consistent data models. Each API endpoint will document exactly the data fields returned and what each field means. Access to the documentation won't require approval. Some exceptions to open documentation may be made for sensitive or protected data.
Figure 3 illustrates the workflow of application registration. It starts by allowing the developer or service owner to go into the API Gateway to access the developer portal using their SSO credentials. Developers will be able to browse a catalog of the available APIs, examples and register their application to request access. Once the application registration is approved, the service owner or developer is granted an API key. This API key is used by the application each time it needs to request information from the API gateway.

Security and Authorization
The developers of this API architecture will work closely with the Office of Information Security to validate and verify the architecture. All the communication and data delivery will be encrypted end-to-end through HTTPS. As part of the development process of the APIs, the development
team will include automated penetration tests through a library of known exploits. The code
developed by the API project will go through code reviews and include automated tests. Tokens
with an expiration of six months will be used to authenticate between systems. The API tokens
will be unique for each integration. Token renewal will be automated to lessen the burden on
application developers. Logs and monitoring will be in place to ensure that requests to a system
can be traced back to the consumer of the data. All the APIs logs will be shipped to a
centralized logging system, such as Splunk. With a centralized logging system, triggers and
alerts can be setup to notify API developers if an error or issue occurs.
As shown in figure 4, systems won't talk directly to the data providers; instead they will go through an API gateway layer. In this figure, we see that the lines for sharing / moving data between systems are greatly simplified. This simplified method of communication, translates to a simpler architecture, and less maintenance costs for integrations. This layer will act as a proxy. This communication can be secured at the network layer by only allowing the API gateway to talk directly to the data provider (see figure 1, data flow). API authentication and authorization is a complex problem. This is an area in which the API project will leverage work done by others in this space. The API gateway layer is a service / appliance that can be provided by a 3rd party or open source. This approach to securing the access to APIs will improve data security and save OSU developer resources. The API Gateway will be hosted within data centers in the United States to meet Export Controlled data regulations. Any cached data by the API Gateway will be encrypted. If a 3rd party vendor is selected for the API Gateway, we'll look for PCI and HIPAA compliance.

**Figure 4 API Gateway Nova**

**IDM and the API Architecture**

The API project and the Identity Management (IDM) group at OSU share some similarities in their problem scope. Authorization is one of IDM's concerns. Currently many applications have data feeds to Banner only to perform user authorization (e.g.: checking if a user is an active
student, or belongs to a specific group or is an international student). Many of these data feed integrations will be replaced with SAML or LDAP attributes. The API project team will work closely to prevent chasing the same problem or reinventing similar solutions.

Some of the current data feeds should be replaced in the future with entitlement and group membership provided via Grouper. Grouper fits with the API architecture philosophy of microservices. Instead of every application at OSU developing and reinventing a solution for authorization, Grouper APIs should be leveraged. Grouper will then become one of the data providers and APIs that the API Gateway will proxy and provide to systems.

One disadvantage of SAML attributes is that they are only provided during the user authentication step. This shifts some responsibility onto the application. If the application is searching for information about the currently logged in user, they can use SAML and cache SAML attributes. If the application needs to access this information after the user has logged in and the information wasn't cached, the application will have to use APIs instead.

**Milestones**

Our development approach will lean towards short frequent iterative milestones. The API architecture will need the agility to respond to feedback quickly, in minutes rather than days or weeks. The frequent release cycle, with a focus on agility, will enable the API project to shift resources as needed. Deployments and releases of APIs will become a known automated process. Smaller achievable successful releases will be the preferred approach versus one huge successful release every couple of years. The first few releases of APIs will focus on solidifying the architecture and web-based API foundation and on commonly requested integrations. After a few releases, the API project will re-assess and prioritize resources to address developers' feedback.

**Outcomes**

The API architecture delivery scope ends at providing as JSON or XML representation of data. The architecture won't focus on providing the same data in a variety of formats, such as CSV, Excel, SQL dump or flat files. Instead, the architecture will emphasize the need for an IS ETL strategy. If a system needs data in a format other than the JSON or XML provided by the APIs, an ETL, provided either by IS or by the department, can massage the data as needed. The community of OSU developers will have access to a developer portal with API documentation, examples and consistent data models. This architecture will pave the path and setup the infrastructure for data interoperability via web APIs. This architecture will demonstrate to the OSU developer community, colleges and departments a sustainable and scalable approach while making it easy for other units on campus to start developing their own APIs, publish their documentation, leverage the API Gateway and authenticate applications in a consistent and secure manner.
Benefits

REST based APIs will be leveraged to provide near real-time integrations between systems. This reliability and resiliency of services will reduce the need for developers at OSU to cache data locally in plain text. Systems won't have to parse a full data dump just to get a few rows of data that they need for processing.

The API layer will strip personal identifiable information (PII) and protected data from the application data by default. By centralizing the PII stripping data process, OSU will realize a consistent approach for isolating PII data from integrations that don't require it. This centralization of efforts will improve security and save OSU's resources. For business use cases that require PII, access to the data will require approval during the application registration process. Protected data will only be provided when a business use case exception grants it. Any protected data fields will be encrypted on top of HTTPS.

Other benefits from this API based architecture include improved implementation / rollout time for new services, reduced load on data stewards, and support open access efforts. In the old integration pattern, getting direct access to the databases used by systems of records was difficult or not possible due to security restrictions. With the new API architecture, getting access to data and managing the security behind access will be streamlined.

Conclusion

The current data interoperability approach consists of low level access and custom data models for each application. As indicated by the lifespan principles in the companion document, Data Interoperability at OSU, architecture must support technology innovation cycles of three to four years while persisting the data for many more years. The API architecture will provide loosely coupled interoperability for a variety of data consumers. The new data interoperability model will provide consistent data models, documentation, improved security and near real time integration.

The data level agreements will streamline the process of granting applications access to the needed data. The motivation behind the new architecture is to enable innovation by improving the integration of new applications. The OSU community should be able to use the best application to perform a task. Departments will benefit from being able to use intuitive, mobile friendly and modern applications. OSU will save money because developers won't have to write code multiple times to provide the same data. All these benefits are provided by the RESTful architecture based on proven standards, protocols, and design patterns.