Ed-Fi REST API
Design & Implementation Guidelines

Ed-Fi Data Standard v2.0 (API Guidelines v1.0)
April 28, 2015

© 2015 Ed-Fi Alliance, LLC. All rights reserved. Ed-Fi is a registered trademark of the Ed-Fi Alliance, LLC.
For the latest information about the Ed-Fi Alliance, visit our website at www.ed-fi.org.
Contents

Introduction .................................................................................................................................1

Audience ..............................................................................................................................1

Prerequisites ......................................................................................................................1

Scope ......................................................................................................................................1

Data Model ..........................................................................................................................2

Architectural Style ...............................................................................................................2

Key Characteristics ...............................................................................................................4

Benefits ...............................................................................................................................4

Security ..................................................................................................................................5

Application Use Cases ........................................................................................................6

Requirement Levels ............................................................................................................7

API Design Guidelines .........................................................................................................8

Resources ..............................................................................................................................8

Resource Extensions ..........................................................................................................10

Types and Descriptors .......................................................................................................10

HTTP Verbs ..........................................................................................................................11

General Request Construction ..........................................................................................12

Resource Collections and Individual Resources .................................................................12

Associated Resources .........................................................................................................12

URI Construction and HTTP Verb Usage for Individual Records and Transactions .............12

Ed-Fi Descriptors ................................................................................................................13

URI Construction and HTTP Verb Usage for Ed-Fi Descriptors ...........................................14

Descriptor References ......................................................................................................14

Bulk Operations ................................................................................................................14

Bulk Operation Recommendations .....................................................................................15

Bulk Packet Format .............................................................................................................15

Query Operators ................................................................................................................16

Search ..................................................................................................................................16

Selectors .............................................................................................................................17
Paging .................................................................................................................................17
Views .................................................................................................................................18
Response Codes ................................................................................................................18
  Required Response Codes .............................................................................................18
Errors ..................................................................................................................................19
ETags ...................................................................................................................................19
Other REST API Conventions and Features .....................................................................20
API Implementation Guidelines .........................................................................................20
  Handling Authentication and Authorization ...................................................................20
    Authentication ..............................................................................................................21
    Authorization ...............................................................................................................22
    Authentication and Authorization Permutations .........................................................25
Handling Non-repudiation .................................................................................................26
Handling Optimistic Concurrency with ETags .................................................................26
Handling Web Cache Validation with ETags ....................................................................28
Implementing Bulk Operations ........................................................................................28
  Security .........................................................................................................................29
  Atomicity of Data ...........................................................................................................29
  Data Ordering ................................................................................................................29
  Scalability .......................................................................................................................29
Introduction

This document provides design guidelines for an Ed-Fi representational state transfer (REST) application programming interface (API). These guidelines describe the properties to which an API specification and related implementation must adhere in order to be considered “aligned” to Ed-Fi technology standards. They are guidelines only. They do not describe a specific implementation or particular API specification. Where helpful, a rationale for specific guidelines is also provided.

Audience

There are two audiences for these guidelines. The primary audience is developers of an Ed-Fi REST API who are creating specifications for API implementations. The secondary audience is application developers who may find the guidelines useful when building client applications that interact with an Ed-Fi REST API.

Prerequisites

To gain the most benefit from these guidelines, you should be familiar with the following knowledge areas:

- Representational state transfer (REST) services, see https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf, Chapter 5
- OAuth2, see https://tools.ietf.org/html/rfc6749
- JavaScript Object Notation (JSON), see http://www.json.org

Scope

The essential features that characterize an Ed-Fi REST API implementation are the data model that serves as its basis and the REST architectural style.
Data Model

The Ed-Fi Unifying Data Model (UDM)\(^1\) provides the basis for the data transferred and manipulated by an Ed-Fi REST API implementation. The Ed-Fi UDM is a structured, conceptual model of common K-12 education data. The model includes entities that are easily recognized by educators and administrators: schools, students, teachers, attendance, grades, assessment results, and many others. These entities contain attributes (i.e., properties) that are also easily recognized. For example, assessment results contain data, such as a score and the date the assessment was administered. The UDM also includes associations (i.e., relationships) between entities, such as the association between students and schools.

REST interfaces are built around Resources that define nouns. In the education domain these nouns include such things as schools, students, and teachers. In the Ed-Fi UDM these nouns have been rigorously defined as “entities,” with specific attributes and associations. Compositions of entities, with their attributes and associations, are called “domain aggregates.” These are identified from the Ed-Fi UDM according to the principles of Domain-Driven Design (DDD).\(^2\) Domain aggregates are the Resources for an Ed-Fi REST API. These concepts are discussed in more detail later in this document.

An Ed-Fi REST API may cover a subset of the full Ed-Fi UDM that is exposed and exchanged in a particular system or implementation. The API need not be implemented for the entire scope of the Ed-Fi UDM in order to be aligned.

Architectural Style

The REST architectural style\(^3\) is a convention-based approach to defining APIs. HTTPS (Hypertext Transfer Protocol Secure), using the HTTP operations (GET, PUT, POST, DELETE, etc.), is used as the application protocol.

---

1. For more information about the Ed-Fi Unifying Data Model, see http://www.ed-fi.org/tech-docs.
REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of Resources. As depicted below, a data store or server-based application implements, exposes, or is wrapped with an Ed-Fi REST API to allow client applications to exchange and manipulate education data.

![Figure 1. Interaction between REST API and client application](image)

APIs can be thought of as a “contract” between data sources and client applications. The underlying platform and application choices are unimportant in terms of this “contract.” An Ed-Fi REST API follows this pattern. An Ed-Fi REST API levies no technical requirements on how data is internally stored or how it is used by client applications. The API must only provide the technical “contract” between a provider of data and its consumer applications, externally representing the exchanged data Resources in a way that is aligned with the Ed-Fi UDM.

The same resource may be represented to different clients using different representations. For example, an API may represent a resource as JSON for an application that is performing transactions, but may use XML to represent the same resource to another application for bulk data export. The representation is a way to externally represent the resource, but is not the resource itself.

There may be circumstances where an Ed-Fi REST API would diverge from a “pure” REST approach to support specific use cases, for example, to support application-specific operations.

While the Ed-Fi XML Data Exchange Framework allows for file-based exchange of education data between systems, an Ed-Fi REST API extends this capability to include real-time and transactional exchanges of information.
Key Characteristics

An Ed-Fi REST API’s key characteristics provide specific benefits to educational organizations. Security and privacy of data and data systems are primary concerns for all implementations and are addressed in an Ed-Fi REST API. The level of abstraction available with an Ed-Fi REST API also allows for a variety of use cases.

Benefits

An Ed-Fi REST API may be used to facilitate flexibility in a variety of situations where different applications and/or data stores need to consume, exchange, or manipulate education data. The major benefits of an Ed-Fi-aligned API are described below:

- **Data store and application agnosticism.** Education organizations gain greater control over their application data infrastructure when they can use a common API that may be implemented or consumed by any number of vendors. Storage engines (implementing an Ed-Fi REST API) may be selected to meet exact availability, distribution, and scalability needs. Applications (consuming an Ed-Fi REST API) all interoperate in real time, on common data, freeing education organizations to select product suites and/or individual applications that target specific user needs.

- **Data migration simplified.** In much the same way that the Ed-Fi XML Data Exchange Framework creates a simpler migration path from one source of education data to another, the Ed-Fi REST API removes the need to migrate the data at all. Client applications that access a data store with an Ed-Fi REST API can be replaced without data migration.

- **Data consistency between applications.** Educational organizations use many applications. Often, these applications use similar core entities such as student, school, and class. When changes are needed for any of these core entities, the entities must be updated in several systems. Inconsistencies often do not become apparent until the data is combined into a central repository for cross-reporting purposes. An Ed-Fi REST API enables a common repository for core entities, so consistency is maintained across applications at all times.
• **Simplified infrastructure.** The IT staff at many educational organizations are overtaxed with ever-increasing system management, desktop support, and reporting requirements. Each additional application and data repository represents additional “surface area” that must be managed, monitored, maintained, and secured. Infrastructure may be simplified by using an Ed-Fi REST API instead of trying to synchronize between proprietary data stores or application-specific APIs.

• **Open infrastructure.** An Ed-Fi REST API is built on current industry best practices and standard HTTP verbs. Therefore, an Ed-Fi REST API neither requires nor precludes cloud-based providers (e.g., data repositories) or consumers (e.g., desktop or mobile applications), or data store topology (relational or document storage). The educational organization has the choice to use any of these technologies.

### Security

Security is necessarily a major concern for all organizations that deal with education data. An Ed-Fi REST API addresses those security concerns in specific ways. Security, in this context, consists primarily of three activities:

- Identifying users and client applications seeking access to information (i.e., authentication)
- Establishing access policies to information (i.e., authorization)
- Enforcing those access policies

An Ed-Fi REST API platform containing personally identifiable data or data about which there are privacy concerns will limit access to authenticated and authorized client applications. Even systems that deal only with public data should secure access by authorizing and authenticating all access.

More details and guidance regarding security are provided in the “API Implementation Design Guidelines” section of this document.
Application Use Cases

An Ed-Fi REST API provides organizations developing systems that exchange education information with a wide variety of possible scenarios. The following examples represent only a sampling of the most compelling use cases:

• **As a shared application data repository.** A state (or large district) can have a combination of extremely large and extremely small schools. A large school often has more specialized roles than a small one. An enterprise Student Information System (SIS) that suits the needs of a large school may be quite different from the SIS appropriate for a smaller school. Using an Ed-Fi REST API, each school can use a SIS that is tailored to their specific needs. State or district users can then generate reports across both schools using a software package that meets its needs without requiring data exports from any school. Each application has direct access to the most current information.

• **As an enabler for “best of breed” applications.** A school district may prefer their SIS for day-to-day use, but need to integrate with data from another system to leverage best-practice, off-the-shelf analytics. With the Ed-Fi REST API, a more capable reporting package can be used to supplement the capabilities of their preferred SIS.

• **As the data foundation for targeted “applets.”** Small, highly focused applications can use an existing Ed-Fi REST API to provide parents, teachers, and administrators with web or smartphone applications that target specific needs. Imagine a smartphone application that a high school principal can use to verify the names and class schedule for a student found “wandering” the halls, or an SMS notification application that informs parents the same day that their child misses a key assignment or examination.

• **As a secure source for research data.** Researchers spend much of their time collecting and standardizing education data in order to perform analysis. One time-consuming aspect of this process is stripping away personal data in order to maintain student privacy. This means that researchers analyze data that is months, or even years, out of date. The Ed-Fi REST API can provide near real-time, de-identified data to researchers in a common, secure format.
• **As a simplified data reporting infrastructure.** School districts spend a significant amount of time collecting and reporting information to their state education agencies (SEAs). Multiple departments within the SEA often request the same information. Each data collection takes time. Using an Ed-Fi REST API, a school district can authorize the SEA to directly access only the specific information it needs, thus reducing the time spent by the school district providing redundant information.

• **As an interface for public information.** Most school districts have websites that list information such as school names, grades served at each school, attendance statistics, bell schedules, and available courses. If their websites used the Ed-Fi REST API as a source for the information, this public information could be provided automatically based on the most current information available—without the need to update web pages manually.

**Requirement Levels**

The sections that follow contain guidelines for Ed-Fi REST API designs and implementations. Certain key terms have specific meanings in these guidelines. From this point forward, key terms are italicized when used with the specific meanings shown below:

• **Must, required, shall.** These terms indicate an absolute requirement for an aligned Ed-Fi REST API implementation.

• **Must not, shall not.** These terms indicate an absolute prohibition for an aligned Ed-Fi REST API implementation.

• **Should, recommended.** These terms indicate that there may be valid reasons to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

• **Should not, not recommended.** These terms indicate that there may be circumstances when a particular behavior is acceptable or useful, but the full implications should be understood and the case carefully weighed before implementing such behavior.

---

4 Language for this section has been adapted for use from [http://www.ietf.org/rfc/rfc2119.txt](http://www.ietf.org/rfc/rfc2119.txt). This document conforms to the guidelines provided there.
• **May, optional.** These terms indicate items that are truly optional. One organization may choose to include the item because their specific implementation requires it or because they feel it enhances the implementation, while another organization may omit the item. Client applications must be prepared to interoperate with an implementation that does not include the option, although perhaps with reduced functionality. In the same vein, a client who does include a particular option must be prepared to interoperate with another implementation that does not include the option.

• **Aligned.** An aligned implementation meets all the must, required, and shall requirements of these guidelines.

## API Design Guidelines

An API is essentially an interface used by programmers to support communication between systems. Guidelines are useful because the REST architectural style allows for a great deal of flexibility in design. The guidelines that follow are designed to make Ed-Fi REST API implementations intuitive and easy to use for programmers.

### Resources

The Resources that are used by an Ed-Fi REST API are compositions of entities, attributes, and associations. As described above, Resources are domain aggregates that have been identified from the Ed-Fi data model according to the principles of Domain-Driven Design (DDD). Use cases and events in the domain typically center on individual domain aggregates.

Domain aggregates are organized along transactional boundaries, where the data contained should “live” and “die” together. For example, the Discipline Incident domain aggregate contains details about a discipline incident, and also captures data related to the behaviors and students involved. A Discipline Action is not generally captured at the same time as the Discipline Incident. Therefore, each is separated into its own domain aggregate.
Each domain aggregate has an “aggregate root.” An aggregate root is an entity (and in some cases an association) that includes other entities, their attributes, and associations. The subordinate entities, attributes, and associations of a domain aggregate are not directly accessible and can only be referenced through the aggregate root.

Most entities in the UDM are aggregate roots (e.g., Student, School, and Course). They contain no other entities. In some cases, an association that represents a significant domain concept is also represented as an aggregate root. For example, the StudentSchoolAssociation is represented as an aggregate root because it reflects enrollment.

In the table below, the domain aggregate for a Course is constructed from a number of course-related entities in the Ed-Fi UDM. A complete list of Resources, Types, and Descriptors is included in the API Resource Listing at http://apidocs.ed-fi.org.

<table>
<thead>
<tr>
<th>Domain Aggregate</th>
<th>Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course</td>
<td>Course</td>
</tr>
<tr>
<td></td>
<td>CourseCompetencyLevel</td>
</tr>
<tr>
<td></td>
<td>CourseGradeLevel</td>
</tr>
<tr>
<td></td>
<td>CourseIdentificationCode</td>
</tr>
<tr>
<td></td>
<td>CourseLearningObjective</td>
</tr>
<tr>
<td></td>
<td>CourseLearningStandard</td>
</tr>
<tr>
<td></td>
<td>CourseLevelCharacteristic</td>
</tr>
</tbody>
</table>

Each resource exposed by an Ed-Fi REST API must be referenced by an internally-assigned universally unique identifier (UUID). While the specific algorithm for generating these identifiers is not prescribed in these guidelines, the identifiers should be generated using a UUID implementation such as Microsoft’s GUID (globally unique identifier)\(^5\). An Ed-Fi REST API should generate unique identifiers for its clients, and should not accept client-generated identifiers when inserting or updating data.

\(^5\) For additional information on UUIDs, see http://en.wikipedia.org/wiki/Globally_unique_identifier.
All Resources must be created with and also be retrievable by one or more externally defined primary key values. Those values must be natural keys of the resource. For example, a Session is uniquely identified by the Session Name, Term, and School Year. Resources must be accessible by primary key values using the standard HTTP GET query string search syntax:

{resourceURI}?keyField1={value1}&keyField2={value2}

PUT, PATCH, and DELETE operations must be identified using their UUID (i.e., {resourceURI}/{id}). PUT, PATCH, and DELETE operations should also be identifiable using their primary key values (natural keys).

Resource Extensions

The Ed-Fi data model can be extended to modify the structure of existing entities or create entirely new entities. In the context of an Ed-Fi REST API, these are called “resource extensions.” Consumers of an Ed-Fi REST API interact with resource extensions just as they interact with other Resources. For example, if the Student resource has been extended in the data model supported by the API platform host, an API consumer requesting a student will receive the extended resource.

The same practices used for native Ed-Fi REST API interchanges and Resources should be used for extensions. Extension Resources and extension entities on Ed-Fi standard interchange schemas should follow the same conventions as outlined in the Ed-Fi Core XML Schema. Extension attributes (that is, new attributes for existing Resources) do not carry a suffix identifying them as an extension, and are indistinguishable from native attributes when viewed in the API.

Types and Descriptors

Types and Descriptors exposed through the API are analogous to enumerations in XML or lookup values in a database. The primary difference is that Types are generally immutable in the context of particular Ed-Fi technology implementations because they are intended to form a relatively stable part of the data model, while Descriptors are elements designed to support change.

---

Accordingly, there are differences in how these similar structures are supported through the API.

Types must be read-only from the API’s perspective (as they are analogous to enumerations).

Descriptors are typically read-only from the API’s perspective; however, an implementation may allow them to be created using POST and updated using PUT.

**HTTP Verbs**

HTTP verbs communicate actions that can be taken against a resource. Depending on the verb, the request can require additional information in the body. The Ed-Fi REST API supports the following verbs:

- **POST** An HTTP POST creates an individual subordinate resource. If successful, the URI to the new resource is returned in the “Location” HTTP header of the response. Performing a POST with identical natural keys to a resource already in the data store must perform an update rather than create a new resource. A POST operation must not allow a desired unique ID (UUID) to be provided to the REST API. POST is a required verb for non-read-only Resources.

- **GET** An HTTP GET returns existing Resources. Providing the natural key or internal UUID on the URL specifies an individual resource, while omitting the natural keys and UUID retrieves the complete set of Resources of the given type. GET must be an idempotent operation. GET is a required verb.

- **PUT** An HTTP PUT must perform an idempotent update of a resource. PUT performs a full replacement of the existing resource with the supplied value. A PUT against a nonexistent resource is an error condition. PUT is a required verb for non-read-only Resources.

- **DELETE** An HTTP DELETE deletes an existing individual resource. DELETE is a required verb for non-read-only Resources.

- **PATCH** An HTTP PATCH performs a partial update on an existing individual resource. For a partial update, only the properties that are submitted will be updated on the target resource. The entire patch will be applied, or
none of it will. The new representation of the entire resource is returned in the response body. PATCH is a recommended verb for non-read-only Resources.

**General Request Construction**

For all Ed-Fi REST API transactional requests and responses, JSON must be the default format. If a media-type header is not provided, “application/json” is presumed. Alternate packet formats may also be supported.

**Resource Collections and Individual Resources**

For each resource, there are two base forms for the URI: one for a collection of Resources and the other for a specific resource in the collection. The collection form for the URI is referred to by the pluralized name of the individual resource. A specific resource is referenced by the collection name, followed by a slash and the resource’s unique identifier. For example:

- /students refers to a collection of students
- /students/ffc0...a272 refers to a specific student with an assigned identifier of “ffc0...a272”

**Associated Resources**

A collection of Resources associated with another resource may be referenced in a single URI, using the pattern associatedResource/{id}/resource. The collection to be returned must be the rightmost resource. For example:

- /schools/abc7...123f/studentSchoolAssociations/students refers to the collection of Students associated with a specific School

**URI Construction and HTTP Verb Usage for Individual Records and Transactions**

Individual record and transaction URIs in an Ed-Fi REST API take a convention-based approach to construction and HTTP verb usage.
Table 2. Example of Convention-Based Approach to Construction

<table>
<thead>
<tr>
<th>Resource</th>
<th>POST</th>
<th>GET</th>
<th>PUT/PATCH</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>/students</td>
<td>Add a new Student</td>
<td>Get a collection of Students</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td>/students/{id}</td>
<td>Error</td>
<td>Gets an individual Student</td>
<td>Updates an individual Student</td>
<td>Deletes an individual Student</td>
</tr>
<tr>
<td>/students/{id}/section</td>
<td>Error</td>
<td>Gets a collection of Sections associated with a specific Student</td>
<td>Error</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Ed-Fi Descriptors**

Descriptors in the Ed-Fi data standard are a set of mechanisms to support flexible enumerations or code tables. Each Descriptor has the following attributes:

- `[abc]DescriptorId` (primary key)\(^7\)
- namespace
- codeValue
- shortDescription
- description
- priorDescriptorId
- effectiveBeginDate
- effectiveEndDate
- A “map” back to an Ed-Fi enumeration value

The GET of a resource *must* return the namespace and codeValue for Descriptor enumerations. Other components of the Descriptor can be retrieved from the Descriptor resource.

PUTs of a resource *must* specify the namespace and codeValue for each Descriptor value.

---

\(^7\) Where `[abc]` is the name of a specific Descriptor.
URI Construction and HTTP Verb Usage for Ed-Fi Descriptors

Descriptors are also exposed as Resources of an Ed-Fi REST API and can be accessed and manipulated as follows:

Table 3. Accessing and Manipulating Descriptors

<table>
<thead>
<tr>
<th>Resource</th>
<th>POST</th>
<th>GET</th>
<th>PUT</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[abc] Descriptors</td>
<td>Adds a new Descriptor</td>
<td>Gets all Descriptors for the subtype</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td>/[abc] Descriptors/{id}</td>
<td>Error</td>
<td>Gets all attributes for an individual Descriptor</td>
<td>Updates an individual Descriptor</td>
<td>Deletes an individual Descriptor</td>
</tr>
</tbody>
</table>

Descriptor References

References to a Descriptor value are a URI constructed with the namespace followed by the URL-encoded shortDescription:

```
[namespace]/[shortDescription]
```

For example, to refer to the behaviorDescriptor value in the Ed-Fi namespace with a short description of “School Violation,” the reference would be the following URI:

```
https://www.ed-fi.org/
Descriptor/BehaviorDescriptor.xml/School%20Violation
```

Bulk Operations

Bulk operations may be implemented for the purpose of moving a large amount of data at one time without the overhead of individual calls for each resource.

Externally administered test scores (for example, SAT, ACT, or statewide standardized tests) are often received as data files (not individual transactions) and thus will require bulk loading. An online assessment product may not support transactional updates through an Ed-Fi REST API from within their software, and will instead find it necessary to supply a bulk upload of data at regular intervals. Even SIS products that do support transactional updates may
still find it useful to do a one-time bulk loading of data when initially connecting to a data store through an API.

Bulk export operations are useful when clients need an efficient way to obtain a large data set from the hosting platform. Export is also useful where clients of the API host need data files to adhere to a particular format (e.g., exchange with another system that is not capable of its own direct exchange with the API host).

**Bulk Operation Recommendations**

When bulk operations are supported through an Ed-Fi REST API, consider the following guidelines:

- **File Management.** Bulk operations work on files that can take a non-trivial amount of time to load or, in the case of an export, for the server to generate. A bulk import operation API *should* allow files to be uploaded incrementally and for file uploads to be resumed.

- **Control Protocol.** Bulk operations implemented as part of an Ed-Fi REST API *should* be managed over HTTPS.

- **File Transfer.** Using HTTPS for file transfer is recommended for consistency with the REST portion of the API as well as with the encryption. Other secure protocols may be used.

- **Status.** Status information *should* be available (pulled via GET) or provided (pushed via POST) regarding the progress and results of batch operations.

**Bulk Packet Format**

While JSON is a common packet format for many web APIs (including the transactional operations of the Ed-Fi ODS API), bulk data transfers between systems are more suited to XML. XML documents have some advantages over JSON documents, foremost among these is the ability to be validated and parsed by native libraries on a variety of platforms. JSON does not include a specification for schema validation. Accordingly, bulk data exchange

---

performed through Ed-Fi REST API operations leverage interchanges built from the Ed-Fi Core XML Schema.

XML documents are considered “well formed” when they conform to the XML specification, they are considered “valid” when they can be validated against a specific XML schema, and they are considered “correct” when the contained information may be correctly interpreted by other systems. For the bulk loading operations of an Ed-Fi REST API, “correct” implies that applicable references to Resources exist at the conclusion of a bulk load operation.

XML is the recommended packet format for the bulk operations of an Ed-Fi REST API. It is further recommended that the Ed-Fi Standard Interchange Schemas be used where appropriate. Where applicable, bulk exports of data should align with bulk loading formats to enable a round trip of information.

**Query Operators**

An Ed-Fi REST API should support searching capabilities with the possible use of selectors, paging, and views. These are discussed below.

**Search**

An Ed-Fi REST API should support querying capabilities when searching a collection of Resources. Query operators are applied to the query string using the following format: `{collectionURI}?{propertyName}{operator}{value}`. Currently, the equals operator is the only operator specified. Other operators (>, <, ‘like’, ‘and’, ‘or’, etc.) may be implemented.

**Table 4. Equals Operator, the Only Operator Specified for Searching a Collection of Resources**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equality</td>
</tr>
</tbody>
</table>

For example, to search all available Students having the first name “John” as an exact match, a `{propertyName}={value}` formulation is used (as shown above):

```
https://api.example.com/v1/students?firstName=john
```
Selectors

Selectors allow application developers to be more selective about how much data is returned in the resource representations. Implementation of selectors in an Ed-Fi REST API is optional.

Table 5. Selectors Allow Increased Selectivity in the Data That Is Returned

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeFields</td>
<td>Limits the response to the fields listed.</td>
</tr>
<tr>
<td>excludeFields</td>
<td>Limits the response to all fields except those listed.</td>
</tr>
</tbody>
</table>

For example, to retrieve only a Student’s first and last names:

https://api.example.com/v1/students/{id}?includeFields=firstName,lastSurname

To retrieve everything except a Student’s middle name:

https://api.example.com/v1/students/{id}?excludeFields=middleName

Paging

Paging is a mechanism that restricts the number of results returned by an operation. Paging is a recommended feature. When multiple records are being returned, the total count of all records must be returned as an additional data member of the response body. The limit parameter may be used in the query string to set the maximum number of records returned. If no value is supplied, the limit parameter should default to 20. The offset parameter may be set to specify how many records to skip when getting the result set. The default value for offset should default to 0.

For example, to get the first name and last name of a collection of available Students from positions 31 to 40:

https://api.example.com/v1/students?fields=firstName,lastSurname&limit=10&offset=30
Views

Views are a recommended feature. Views provide the ability for a client to request predefined and/or custom views be returned as a resource.

For example, a view could include student information, assessments, attendance records, gradebook entries, and transcript information.

Response Codes

REST APIs use HTTP response codes to communicate status information. An API consumer should be able to inspect the HTTP response code and understand the status of its request.

Required Response Codes

The following response codes must be used when responding to requests.

Table 6. Required Response Codes

<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>Name</th>
<th>Reason(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>Returned after a successful operation when a response contains a body.</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>Returned after a successful POST. The response from a POST will also include a Location in the Header pointing to the newly added resource. A POST response will not contain a body.</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
<td>Returned when the server has fulfilled the request, but does not return an entity body.</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
<td>Returned when the client includes the “If-None-Match” header containing the requested resource’s last known Entity Tag.</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>Returned if the request is malformed. The body of the response may contain a descriptive error message.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>Returned if the access token is invalid. The response will not contain a body.</td>
</tr>
</tbody>
</table>
### HTTP Response Codes

<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>Name</th>
<th>Reason(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>Returned when the server is refusing to fulfill a request in situations such as:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Resource fails data validation (missing required properties, string lengths too long, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Resource fails uniqueness validation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HTTP headers required for the operation were not present (e.g., Missing “If-Match” header on PATCH or DELETE)</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>Returned if a resource is not found. The response will not contain a body.</td>
</tr>
<tr>
<td>409</td>
<td>Conflict</td>
<td>Returned when there is any type of referential integrity violation.</td>
</tr>
<tr>
<td>412</td>
<td>Precondition Failed</td>
<td>Returned if an “If-Match” header pre-condition fails.</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
<td>Returned if the server encountered an unexpected error during the operation.</td>
</tr>
</tbody>
</table>

### Errors

If an error occurs on the server, a 500 (Internal Server Error) code must be returned. A message in the body, containing the error details, is optional.

For example:

```json
{
    "code": 500,
    "type": "Internal Server Error",
    "message": "Unable to communicate with database"
}
```

### ETags

ETags (Entity Tags) are mechanisms used to support optimistic concurrency and efficient bandwidth handling. The use of ETags is recommended for Ed-Fi REST API implementations.

---

Other REST API Conventions and Features

Three additional REST API features—case sensitivity, encryption, and version—are discussed in the following table.

Table 7. Additional REST API Features

<table>
<thead>
<tr>
<th>REST Feature</th>
<th>Ed-Fi Implementation</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| Case Sensitivity | /users?firstName=JOHN  
/users?FIRSTNAME=John |   URIs, parameter names, and parameter values must not be case sensitive. The two URI’s to the left will produce the same results. |
| Encryption    | HTTPS                                        | All calls to the API must use SSL.                                          |
| Version       | https://api.example.com/v1/users  
or ...  
https://example.com/api/v1/users | The version number should be specified in the base URI.                      |

API Implementation Guidelines

Ed-Fi REST API implementations typically require features for authentication and authorization, non-repudiation, optimistic concurrency, web cache validation, and bulk operations. Guidelines for implementation of those features are provided below.

Handling Authentication and Authorization

Because security is an ongoing concern and best security practices are continually evolving, an Ed-Fi REST API implementation should not preclude any future security enhancements. Implementations should reflect the current best practices where they affect the actual API structure or use. Additional layers of security may be employed on top of those identified here.

In order to secure a data repository, the repository must know who is requesting an operation, and what the requestor is allowed to do. The question of “who” is known as authentication; the “what” is authorization.
Authentication

The process of authentication provides the means of reliably identifying applications and users to an Ed-Fi REST API implementation. Both applications and users require authentication in order to maintain secure data. The OAuth 2 specification\(^{10}\) is used by an Ed-Fi REST API for authentication. This specification is broad enough to handle both private and public identity providers. An identity provider issues identification information for all providers desiring interaction with the system. An authentication module verifies a security token as an alternative to explicitly authenticating the user.

Both application authentication and user authentication should be used. When applications do not have user authentication associated with API calls, the API must allow application-only authentication. When user credentials are available, the API must require both application and user credentials. These situations are described below.

Application Authentication

In an Ed-Fi REST API, an application is identified when it provides credentials and receives a token that is subsequently used for each interaction during a given period of time (often called a session). An Ed-Fi REST API should provide a method of application authentication. Two-legged OAuth is the recommended implementation for application authentication.

Applications accessing an Ed-Fi REST API should be fully vetted to perform as expected before production application credentials are issued.

User Authentication

Users are identified to an Ed-Fi REST API by one or more trusted Security Token Services (STS)\(^{11}\). The trust relationship between the API and STS is established when the API is configured. The STS may use any means of identification acceptable to the education organization. Three-legged OAuth between the STS and API is the currently recommended implementation for user authentication.


Three-legged OAuth allows for many authentication schemes. Some examples of user authentication schemes include: username and password, client certificates, biometric identification, smart card, and multi-factor authentication. Multi-factor authentication requires that a user have one or more additional pieces of information provided out-of-band to another device such as a mobile phone (often via text message) or generated by an external application using a personal time-based random sequence.

**User Claims**

A claim is a piece of information about a user provided by an STS to an application. Typically, this claim establishes the identity of the user. Examples of claims include: credential (login name), credential originator, email address, display name, and identification key(s) or number(s). Other times, a claim may resemble a role; it is not uncommon to see “job title,” “department,” or “organization” claims.

Claims from the STS should be used to identify a user. Claims may also be used (directly or indirectly) to provide authorization clues to an Ed-Fi REST API.

**System Applications**

Some classes of applications do not naturally operate in the context of a user; they do not have user authentication associated with API calls. Examples of this type of application include: bulk load and extract tools, enterprise-wide SISs, and some classes of reporting tools. In these cases, extreme discretion should be used when considering eliminating user authentication from the security framework for these implementation scenarios.

In these cases it becomes necessary for the API to allow application-only authentication or, preferably, to operate as a “system” user with global privileges.

**Authorization**

Authorization is a set of mechanisms for identifying what operations can be performed, and upon which Resources. Due to privacy concerns and FERPA (the Family Educational Rights and Privacy Act) regulations, it is critical that an Ed-Fi REST API implementation correctly authorizes all resource requests.
The principle of least privilege should be used for an Ed-Fi REST API implementation. Least privilege means that default application and user permissions are “no privileges” and that all privileges be explicitly granted.

Privileges for applications and users should be assigned out-of-band to an Ed-Fi REST API.

**Application Authorization**

Applications perform actions against data based on user directives. It is important to consider what actions are appropriate in relation to specific data. In some cases, it may not be appropriate for an application to act within a specific resource domain. For example, an application that only records attendance should not have authorization to access assessment information. In the same way, it may not be appropriate for an application to be authorized to act within all resource domains. For example, a SIS package used only at one school should not be authorized to work with Resources at other schools.

Application credentials should be assigned out-of-band for each resource group and (if necessary) each organization appropriate to the application within an Ed-Fi REST API.

**User Authorization**

User authorization typically consists of a subset of actions available to an application. However, users may be authorized to perform different actions on different classes of objects, or on distinct objects (domains). An Ed-Fi REST API should provide user authorization in a manner consistent with the sensitivity of the data provided.

**User Roles**

A “role” is a set of operations able to be performed by a user. Roles are typically broad-grained, cross-cutting, and application-specific. A “Teacher” role identifies a set of operations available to an individual, but does not intrinsically limit those operations to a specific classroom.

The specific operations performed by a role such as a “Teacher” will be different from one application to the next. Thus, an Ed-Fi REST API implementation should support resource-oriented claims issued based on the meaning of a
given role in the implementation’s context. This allows security to be managed at a conceptual level (e.g., by assigning personnel to conceptual roles like “Teacher” or “Principal”) while also supporting a fine-grained assignment of permissions to those roles specific to an organization or application.

External User Authorization

Many existing external applications (like SISs) have business logic that limit the operations available to users based on the users’ roles as well as the resource upon which the user is operating. In this scenario, the Ed-Fi REST API provides application authorization, and the client application provides user authorization.

In a two-legged OAuth scenario with a trusted partner application (such as a SIS), resource requests should be secured by scoping requests to specific education organizations such as states, regional service centers, local education agencies, or schools. Within the authorized scope, the applications would be able to access and modify all appropriate data.

Internal User Authorization

An Ed-Fi REST API implementation limits the operations available to users based on the user’s roles and Resources upon which the user is operating. Internal user authorization is preferred over external user authorization because it ensures a consistent security model across all applications and simplifies the application vetting process.

In order to communicate visual cues to users regarding their authorization, user authorization must be communicated to applications. Authentication claims may be enhanced to reflect high-level application-specific permissions using an out-of-band permission look-up. In an implementation of an Ed-Fi REST API, the HTTP OPTIONS method should be implemented to provide resource-specific authorization information to applications. In any case, where operations are not allowed on a resource, the “405 Method not Allowed” response should be issued for unauthorized requests if the application and user authentication is otherwise valid.

---

12 For more information on the HTTP OPTIONS method, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html (Section 9.2).
Client Applications

For three-legged OAuth scenarios, each request should be authorized based on claims-based security (see below) and Ed-Fi domain data to identify the students for which they have responsibility. For example, superintendents would be granted access to student data for all students in their districts, principals would be granted access to all students in their schools, and teachers would be granted access only to students enrolled in their sections.

Authentication and Authorization Permutations

Authorization presupposes authentication. In the figure below, the empty boxes represent impossible security models. The lightest boxes (on the upper left) are less secure combinations and should not be used. The medium blue boxes are security models requiring that applications be certified for their intended purposes and should be used only with extreme caution. The darkest box (on the lower right) represents the recommended Ed-Fi REST API Authorization and Authentication model in which both the user and application are authenticated and authorized within the implementation.

![Figure 2. Permutations of Authentication and Authorization](image-url)
Handling Non-repudiation

Where data security is important, an action performed by a user must have an authentication that can be assured to be genuine with a high degree of confidence. This is known as non-repudiation. Once a security environment has been established, operational logs consisting of (minimally) the user, application, resource, operation, and date/time information should be maintained to establish a basis for non-repudiation within an Ed-Fi REST API implementation. These logs should be audited on a regular basis.

Handling Optimistic Concurrency with ETags

Concurrency becomes an issue in high-volume systems with multiple clients accessing the same data. An Ed-Fi REST API can support an “opt-in” optimistic concurrency model using ETags. During PUT and DELETE operations, the API will verify that the resource has not been modified by another party since it was last obtained by the client. If the resource has not changed, the operation will continue normally. If, however, the resource has changed, clients will receive an error as notification that they must obtain the latest version of the resource before attempting further modifications. This approach can be used to prevent “last-in-wins” update scenarios and related potential data loss.

When ETag-enabled systems respond to a GET request for an individual resource (for example, /students/{id}) the response header returned by the API must contain an ETag that uniquely identifies the version of the resource.

The following is an example response header:

```
ETag: "-8588261538364775808"
Content-Type: application/json; charset=utf-8
Cache-Control: private
Content-Length: 1398
```

---


14 When all REST API actions are secure and logged, the user purported to have performed an action must actually have done it. Without appropriate security or logging, it cannot be guaranteed that a specific user actually performed an action on the system.

15 See, for example, [http://en.wikipedia.org/wiki/Optimistic_concurrency_control](http://en.wikipedia.org/wiki/Optimistic_concurrency_control) for a definition and links to further reading.
The following is the body in a GET response corresponding to the example header above:

```json
{
    "schoolId":12345,
    "classPeriodName":"4th Period",
    "classroomIdentificationCode":"abcde",
    "localCourseCode":"Math 101",
    "termTypeId":1,
    "schoolYear":2012,
    "uniqueSectionCode":"3FJ56",
    "sequenceOfCourse":1,
    "availableCredit":1.5
}
```

To opt-in to an optimistic update, the ETag value is added to an “If-Match” header of a subsequent PUT or DELETE request, and the operation will be processed only if the “If-Match” header value matches the latest ETag for the resource stored on the server.

If the ETags do not match, a 412 (Precondition Failed) response code will be returned. If the “If-Match” header is not specified in the request, then the operation must be processed and the server must respond with a response code of 204 (No Content) if the operation succeeds. However, the API may be implemented to require optimistic concurrency for updates and deletes, and if no “If-Match” request header is supplied by the client, it may respond with a general 400 error status code.

For example, here is a header value in a PUT or DELETE request:

```
If-Match: -85898261538364775808
```

The ETag may be generated as a hashed representation of the resource, a version number, a timestamp representing the last modification to the resource, or a unique identifier that is refreshed after each modification to the resource.
Handling Web Cache Validation with ETags

ETags may be used to reduce bandwidth usage by preventing the contents of an unmodified resource from being returned. An Ed-Fi REST API should support such cache validation through the use of the “If-None-Match” request header. If the ETag value supplied in the request header still matches the existing resource, the API may respond with a 304 (Not Modified) status code with no response body, rather than a 200 (OK) with the resource content.

Implementing Bulk Operations

A REST API is one mechanism for bulk data loading into a data store. While other possibilities abound, when a REST API bulk load API is provided, the considerations in the following sections apply.

Security

The bulk data portion of the Ed-Fi REST API may use system user authentication, but must use application authentication. In addition, installations should use one or more additional authentication mechanisms (such as client IP address restrictions, pre-shared certificates, or VPN access) to further secure the bulk data transfer APIs where possible.

Atomicity of Data

REST APIs are based on simple HTTP verbs and use the HTTP protocol for data transfer. The HTTP PUT and POST verbs packet sizes can be limited by the web server and client configurations. There is no size limit in the HTTP protocol itself for PUT or POST operations, however practical limitations exist.\footnote{Internet Information Server and Apache have a default file upload limit of 2GB. Many browsers and frameworks often have this value as a configurable limit as well.}

Packet sizes should be limited to a reasonable maximum based on the capabilities of the host system. All resource transactions are considered “upserts”\footnote{A term representing a combination of an UPDATE (to existing records) and INSERT (for new records) functionality, detailed here: \url{http://en.wikipedia.org/wiki/Merge_(SQL)}.} and are idempotent; a single resource failure within a packet does not invalidate the packet, but subsequent Resources from that packet are processed. Failed uploads may be resubmitted, but are treated like a new submission.
Data Ordering

The potentially large quantity of transactions contained in a batch operation introduces a design trade-off between reduced system responsiveness while the batch operations are run and transactional operations are halted, or transactional operations becoming intermingled with batch transactions.

The selection of a batch load strategy is an implementation concern. Either approach has merit under appropriate circumstances.

Scalability

Any API has the potential to be overwhelmed with operation requests where the API depends upon data- or calculation-intensive resources. Relational data stores, and most especially the referential integrity checks generally employed in their designs, are data-intensive operations. Likewise, the other ACID (Atomicity, Consistency, Isolation, Durability) properties of a relational data store transaction impose computation and data requirements that potentially monopolize the repository resources, leading to intrinsic scalability concerns.

While NoSQL data stores do not have the same intrinsic design constraints, the relational nature of the Resources in the Ed-Fi data model means that incoming data must be verified for referential integrity before the data is accepted as correct.

Whether relational integrity is verified using relational schema or programmatically during bulk data ingestion, this process can be resource-intensive in production systems. Bulk data packages should be accepted for processing after a validity check (which may include an internal referential integrity check) and the submitter should be notified asynchronously when the bulk data has been processed. If the bulk data fails the validity check, the submitter should be notified asynchronously.

The batch should be processed sequentially. Any operation failures within the batch should not halt processing of valid operations. The submitter should be notified of any operation failures asynchronously.

The exact mechanisms for queuing are left to the implementation, but should ensure transactional integrity, first-in/first-out (FIFO) ordering, and minimize long connections and resource locks.