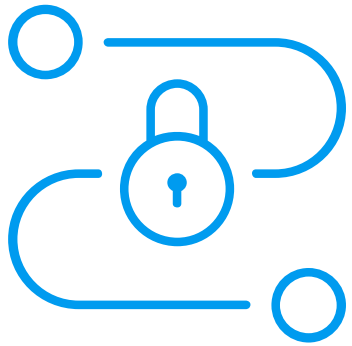
Design and Development of Enterprise grade OAI First JavaScript Microservice



@sumantobiswas, @pramodvallanur

What's the Interface you implemented against?

REST Interface generated from Code

Interface lags Implementation

REST design an afterthought

OAI First

What kind of instance permission does this API need in addition to OAuth scopes?

Behavior deep inside Code

End user concerns like security buried deep inside Implementation

Header, Query parameters, Schemas validation

OAI Driven Behavior

This API's query parameter is camel-cased, whereas the other API accepted dash-cased?

Style inconsistency

Maintenance nightmare

Payload and schema inconsistency

OAI Linting

The OAI Document doesn’t even load in my browser anymore?

OAI document becomes enormous and too verbose

No one wants read OAI anymore

Linting is playing catchup with OAI Style issues

OAI Generation from Metadata

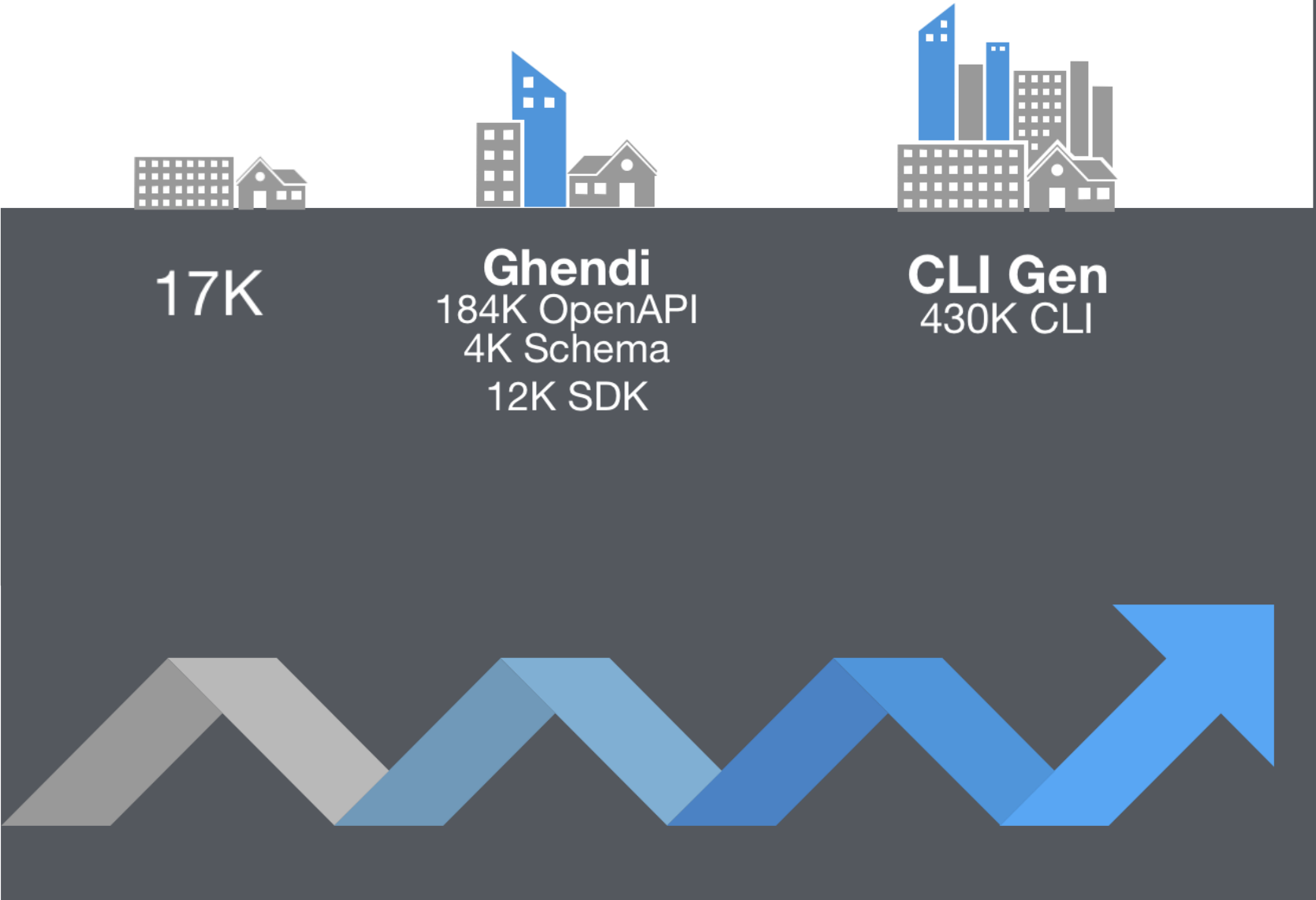
How do I call the REST API from my NodeJS program without having to become a OAI expert?

What about REST Documentation

What about Client Programs?

How do I keep my CLI up to date with REST API

OAI Driven Assets



Highly leveraged DRY resource model
Devops Enabled
100% REST Coverage 100% CLI Coverage
100+ REST Resources
1300+ REST Operations
Perfectly consistent best practice based resource types and REST runtime behavior
Basic CRUD REST resources require zero lines of code
Provider and consumer SDK generation
Auto-generated CLI & REST reference documentation
Auto-generated schema definitions
Tight change governance
REST Style Linting (Kona)
Significantly lowers development, testing, and service costs