OpenAPI Tutorial

29 May 2018

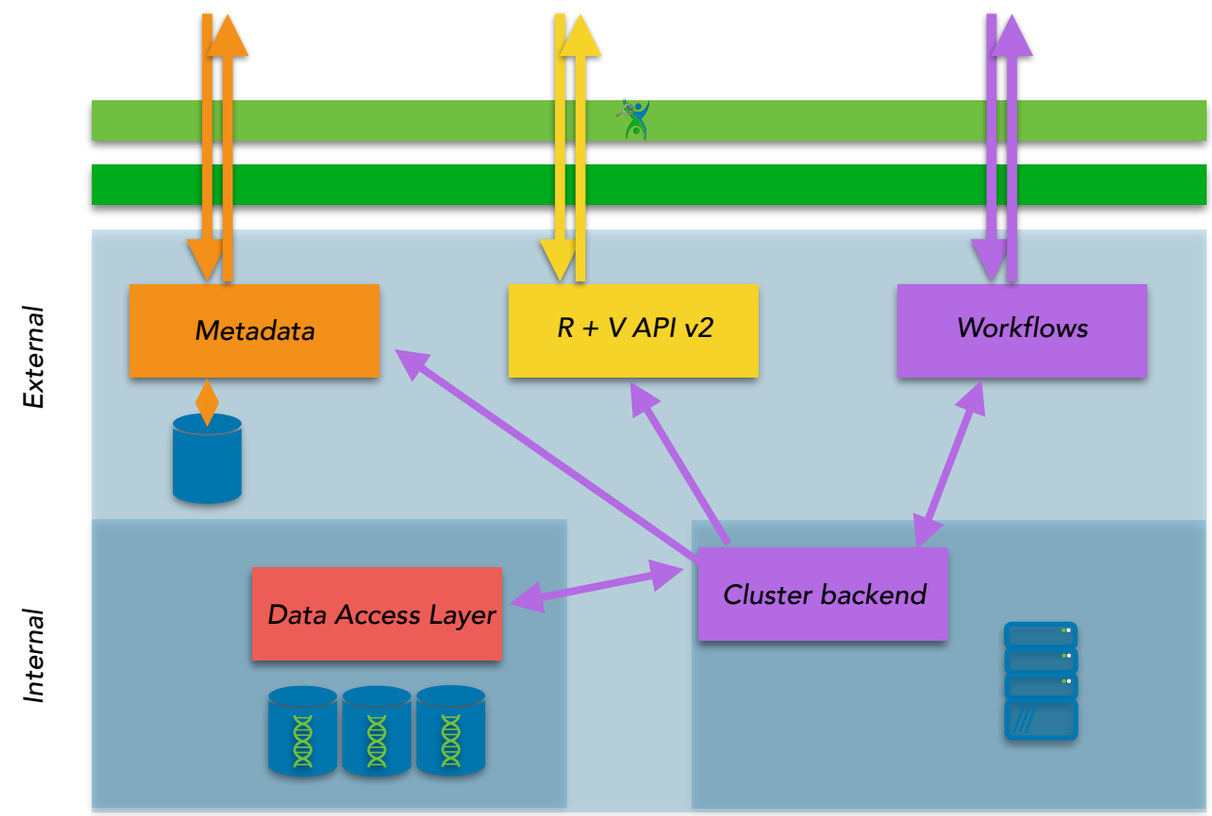
https://github.com/ljdursi/openapi_calls_example

CanDIG

National Analysis of Distributed Private Genomic Data

API-First Development

- All of the tools we're going to be building interact through their APIs:
  - Directly to client
  - To other services
- Makes sense to put hard work of API design up front, work backwards from there
- Otherwise, API tends to reflect **internal implementation** and not **external needs**



API-First Development

- This is especially true in areas where we want to use an external standard
  - Or have something become a standard
- Interoperability means *precisely* defining/ meeting API

Global Alliance
for Genomics & Health
Collaborate. Innovate. Accelerate.

API-First Toolkits

- Two popular API-First toolkits, aimed at different purposes
  - OpenAPI (previously Swagger) - RESTful APIs, typically JSON exchange
  - Protobuf - targeting gRPC (Google's RPC layer)

API-First Toolkits

- Tooling includes:
  - Input/Output validation (types, etc)
  - Mocks (for testing/ developing against)
  - Testing

API-First Toolkits

- In both cases, from the API definition, tools can autogenerate the boilerplate for server and clients
  - In multiple languages
- Just fill in the part that actually does the real work

API-First Toolkits

- The two “API languages” are similar enough that there are tools for converting back and forth
- 90% of things work fine, the edge cases (around defaults, etc) are the tricky parts

API-First Toolkits

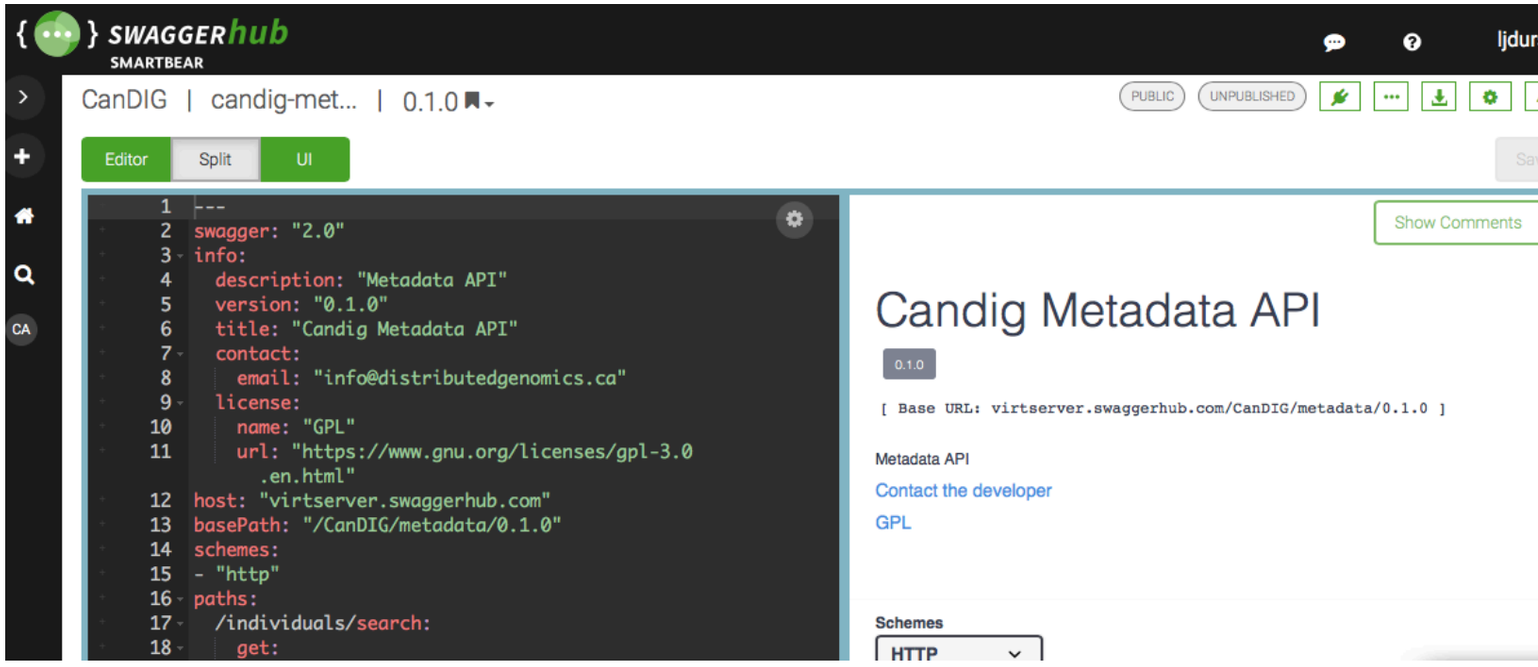
- To target REST APIs for external consumption, OpenAPI is clear winner
  - Protobuf requires another server (gRPC/REST gateway) per service
  - Can write an REST service using Protobuf but requires writing all the boilerplate yourself
  - Kind of defeats purpose

API-First Toolkits

- For internal services, might make sense to use protobuf
- Tooling isn't as flexible, but produces much smaller messages, faster

SwaggerHub

- SwaggerHub.com
- Online examples
- Syntax checker
- Validator
- Will run stubs
- Can export client or server code



Getting started w/ OpenAPI

openapi_calls_example/python/step0/swagger.yaml

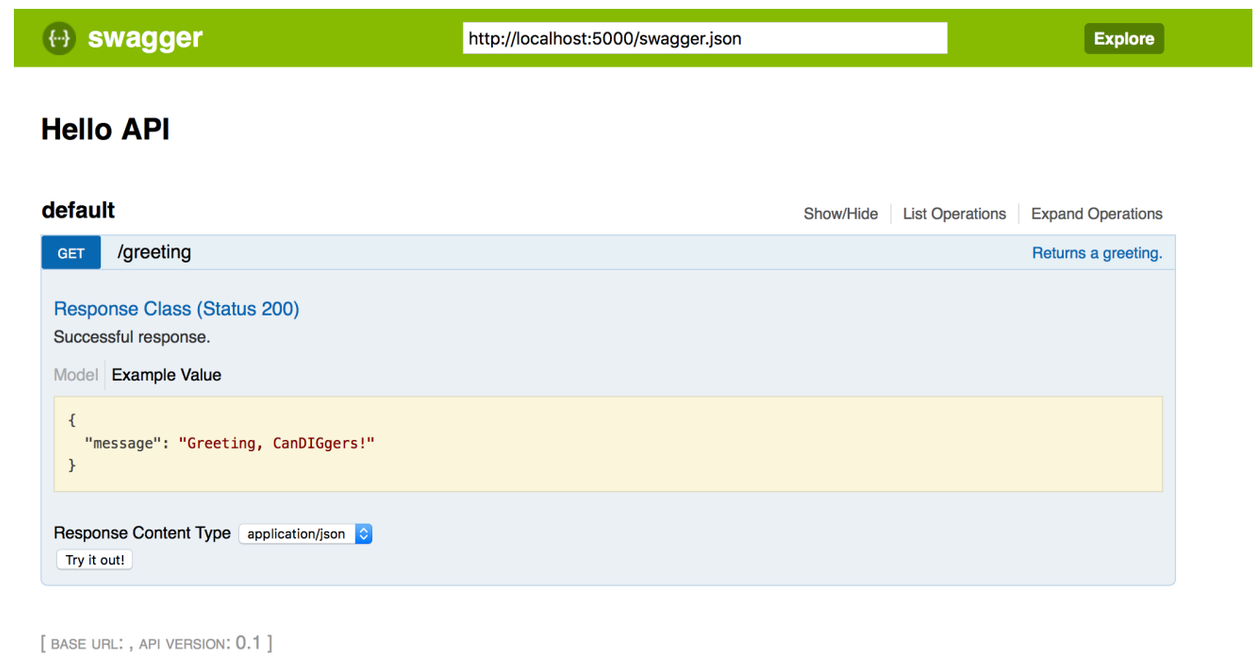
```yaml
swagger: '2.0'
info:
  title: Hello API
  version: "0.1"
paths:
  /greeting:
    get:
      operationId: main.say_hello
      summary: Returns a greeting.
      responses:
        200:
          description: Successful response.
          schema:
            type: object
            properties:
              message:
                type: string
                description: Message greeting
                example: Greeting, CanDIGgers!
```

Getting started w/ OpenAPI

- We'll use Connexion on the python side to convert this into a working Python server
- https://github.com/zalando/connexion
- `pip install -r ../requirements.txt`
- `connexion run swagger.yaml --stub —debug`

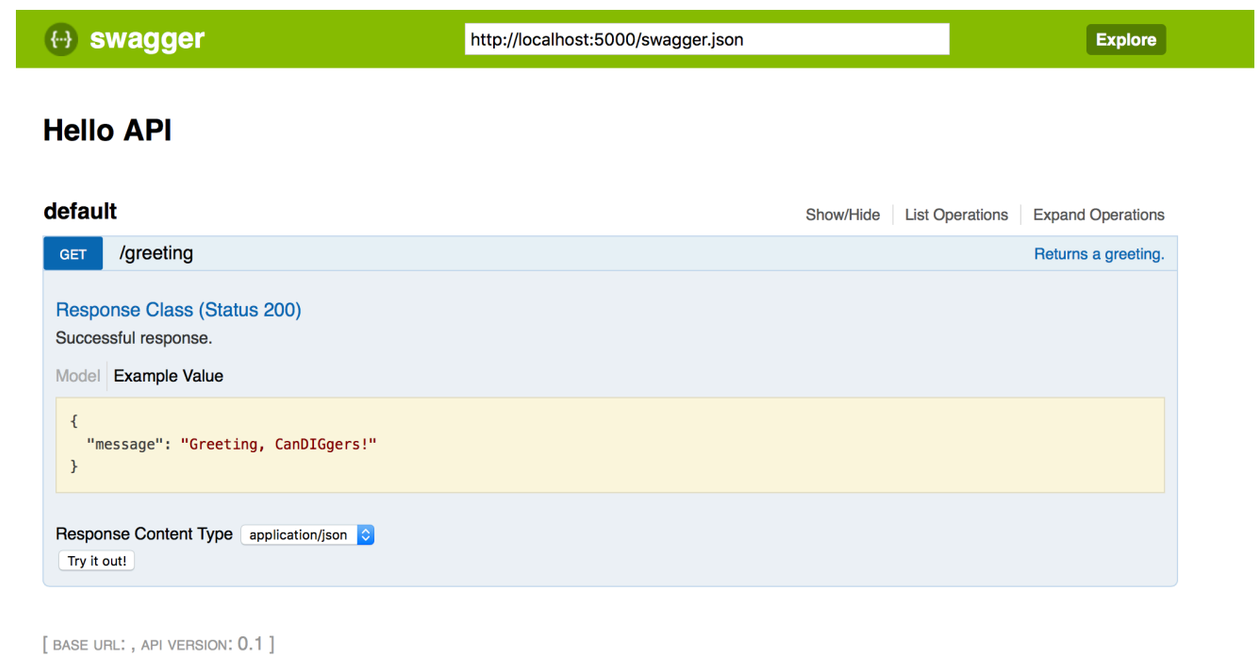
Getting started w/ OpenAPI

- We'll use Connexion on the python side to convert this into a working Python server
- https://github.com/zalando/connexion
- `pip install -r ../requirements.txt`
- `connexion run swagger.yaml --stub —debug`



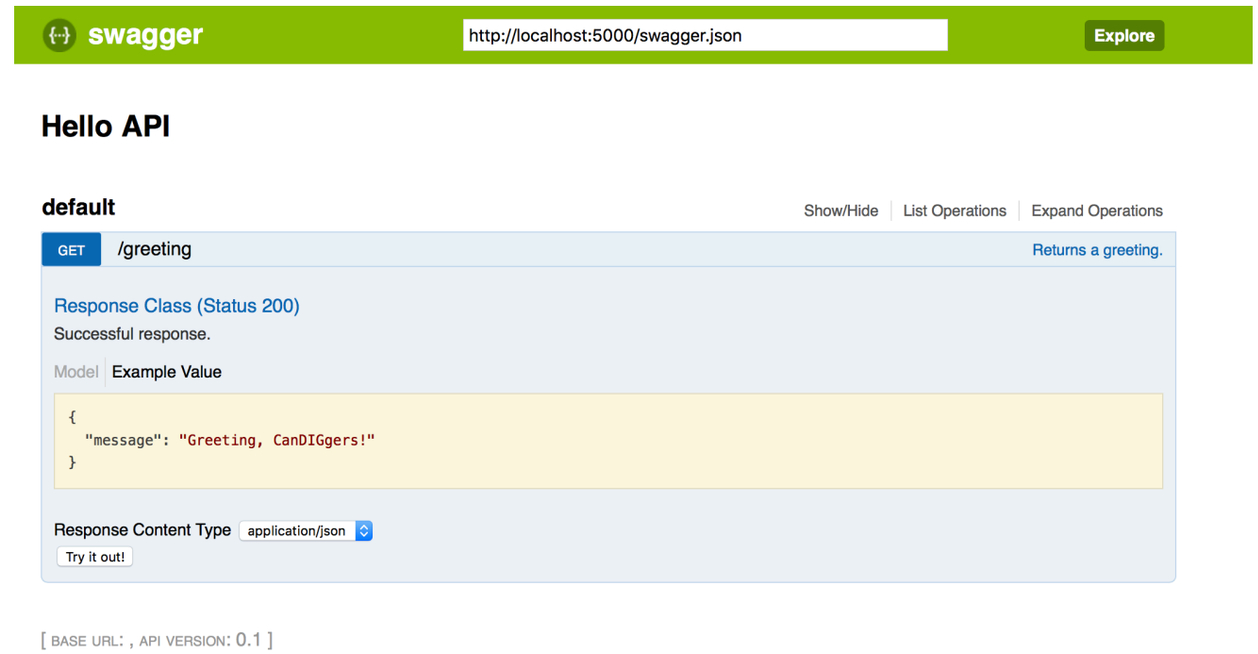
Getting started w/ OpenAPI

- A UI page is shown at /ui
- Can explore the information in the swagger file in nicely-formatted ways
- Can even try the API, with the “Try it out!” button or via curl:



Getting started w/ OpenAPI

- A UI page is shown at /ui
- Can explore the information in the swagger file in nicely-formatted ways
- Can even try the API, with the “Try it out!” button or via curl:
- In “stub” mode, API just returns the example values where available




```
$ curl http://localhost:5000/greeting
{
  "message": "Hello API!"
}
```

Getting started w/ OpenAPI

- However, we can implement the operation as well
- main.py is two lines
- Running without the stub option provides the implemented version
- No boilerplate!

```yaml
swagger: '2.0'
info:
  title: Hello API
  version: "0.1"
paths:
  /greeting:
    get:
      operationId: main.say_hello
      summary: Returns a greeting.
      responses:
        200:
          description: Successful response.
          schema:
            type: object
            properties:
              message:
                type: string
                description: Message greeting
                example: Greeting, CanDIGgers!
```

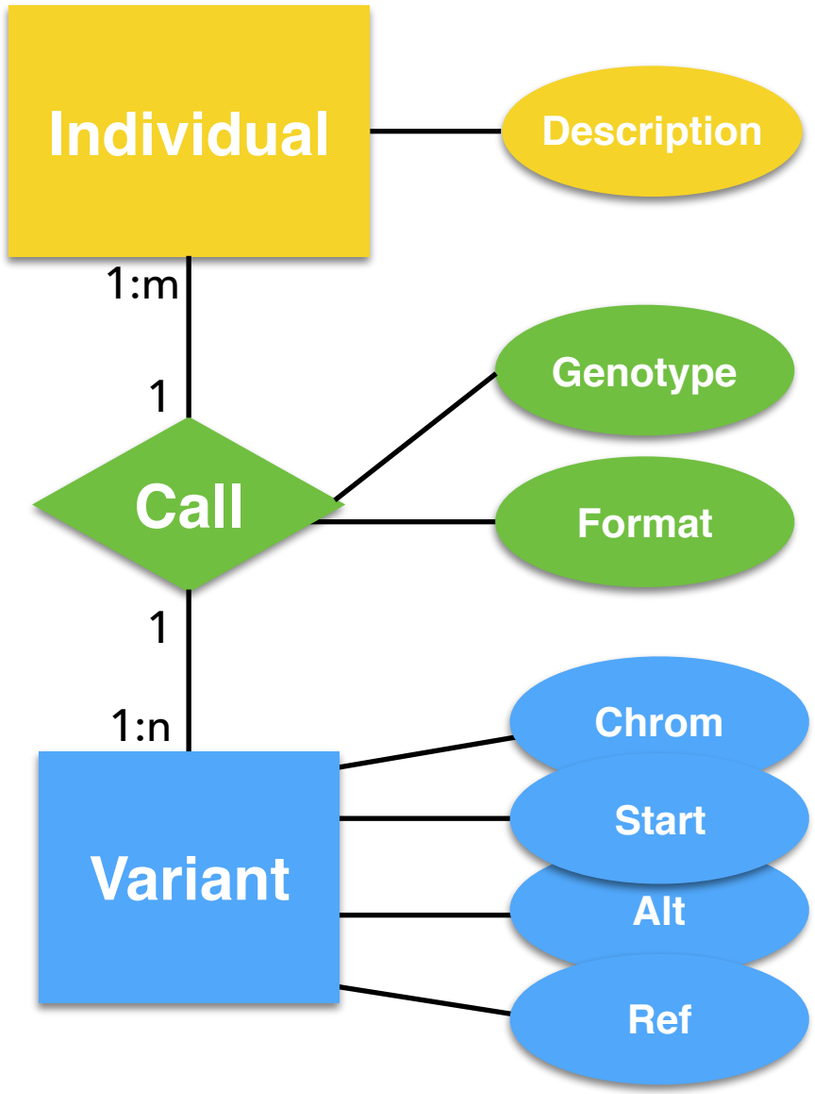
swagger.yaml

```
def say_hello():
    return {"message": "Hello API!"}
```

main.py

Calls, Variants, & Individuals

- Let's take a look at a more complicated case
  - Individuals (really, analyses of samples)
  - Variants
  - Calls (relationship between variant & individual)



Step1

- Let's take a look at the API: definitions

```yaml
definitions:
  Individual:
    type: object
    required:
      - description
    properties:
      id:
        type: integer
        description: Unique identifi
        example: 123
        readOnly: true
      description:
        type: string
        description: description of
        example: "Subject 17"
        minLength: 1
        maxLength: 100
      created:
        type: string
        format: date-time
        description: Creation time
        example: "2015-07-07T15:49:5
        readOnly: true
```

```yaml
Call:
  type: object
  required:
    - individual_id
    - variant_id
    - genotype
  properties:
    id:
      type: integer
      description: Unique identifier
      example: 123
      readOnly: true
    individual_id:
      type: integer
      description: Unique identifier of individua
      example: 123
    variant_id:
      type: integer
      description: Unique identifier of variant
      example: 123
    genotype:
      type: string
      description: Called genotype
      example: "0/1"
    format:
      type: string
      description: Additional call information
      example: "GQ:DP:HQ 48:1:51,51"
      minLength: 0
      maxLength: 100
      default: ""
    created:
      type: string
      format: date-time
      description: Creation time
      example: "2015-07-07T15:49:51.230+02:00"
      readOnly: true
```

```yaml
Variant:
  type: object
  required:
    - chromosome
    - start
    - ref
    - alt
  properties:
    id:
      type: integer
      description: Unique identifier
      example: 123
      readOnly: true
    name:
      type: string
      description: variant name if any
      example: "rs6054257"
      minLength: 0
      maxLength: 100
      default: ""
    chromosome:
      type: string
      description: Chromosome variant occurs on
      example: "chr1"
      minLength: 1
      maxLength: 10
    start:
      type: integer
      description: Beginning location of variant
      example: 14370
    ref:
      type: string
      description: Reference bases
      example: "G"
    alt:
      type: string
      description: Alternate (variant) vases
      example: "A"
```

CanDIG

Step1

- Note! Danger of API-first model; don't necessarily want your internal representation to be completely determined by the API

```yaml
definitions:
  Individual:
    type: object
    required:
      - description
    properties:
      id:
        type: integer
        description: Unique identifi
        example: 123
        readOnly: true
      description:
        type: string
        description: description of
        example: "Subject 17"
        minLength: 1
        maxLength: 100
      created:
        type: string
        format: date-time
        description: Creation time
        example: "2015-07-07T15:49:5
        readOnly: true
```

```yaml
Call:
  type: object
  required:
    - individual_id
    - variant_id
    - genotype
  properties:
    id:
      type: integer
      description: Unique identifier
      example: 123
      readOnly: true
    individual_id:
      type: integer
      description: Unique identifier of individua
      example: 123
    variant_id:
      type: integer
      description: Unique identifier of variant
      example: 123
    genotype:
      type: string
      description: Called genotype
      example: "0/1"
    format:
      type: string
      description: Additional call information
      example: "GQ:DP:HQ 48:1:51,51"
      minLength: 0
      maxLength: 100
      default: ""
    created:
      type: string
      format: date-time
      description: Creation time
      example: "2015-07-07T15:49:51.230+02:00"
      readOnly: true
```

```yaml
Variant:
  type: object
  required:
    - chromosome
    - start
    - ref
    - alt
  properties:
    id:
      type: integer
      description: Unique identifier
      example: 123
      readOnly: true
    name:
      type: string
      description: variant name if any
      example: "rs6054257"
      minLength: 0
      maxLength: 100
      default: ""
    chromosome:
      type: string
      description: Chromosome variant occurs on
      example: "chr1"
      minLength: 1
      maxLength: 10
    start:
      type: integer
      description: Beginning location of variant
      example: 14370
    ref:
      type: string
      description: Reference bases
      example: "G"
    alt:
      type: string
      description: Alternate (variant) vases
      example: "A"
```

CanDIG

Step1

- Note! Danger of API-first model; don't necessarily want your internal representation to be completely determined by the API

```yaml
definitions:
  Individual:
    type: object
    required:
      - description
    properties:
      id:
        type: integer
        description: Unique identifi
        example: 123
        readOnly: true
      description:
        type: string
        description: description of
        example: "Subject 17"
        minLength: 1
        maxLength: 100
      created:
        type: string
        format: date-time
        description: Creation time
        example: "2015-07-07T15:49:5
        readOnly: true
```

```yaml
Call:
  type: object
  required:
    - individual_id
    - variant_id
    - genotype
  properties:
    id:
      type: integer
      description: Unique identifier
      example: 123
      readOnly: true
    individual_id:
      type: integer
      description: Unique identifier of individua
      example: 123
    variant_id:
      type: integer
      description: Unique identifier of variant
      example: 123
    genotype:
      type: string
      description: Called genotype
      example: "0/1"
    format:
      type: string
      description: Additional call information
      example: "GQ:DP:HQ 48:1:51,51"
      minLength: 0
      maxLength: 100
      default: ""
    created:
      type: string
      format: date-time
      description: Creation time
      example: "2015-07-07T15:49:51.230+02:00"
      readOnly: true
```

```yaml
Variant:
  type: object
  required:
    - chromosome
    - start
    - ref
    - alt
  properties:
    id:
      type: integer
      description: Unique identifier
      example: 123
      readOnly: true
    name:
      type: string
      description: variant name if any
      example: "rs6054257"
      minLength: 0
      maxLength: 100
      default: ""
    chromosome:
      type: string
      description: Chromosome variant occurs on
      example: "chr1"
      minLength: 1
      maxLength: 10
    start:
      type: integer
      description: Beginning location of variant
      example: 14370
    ref:
      type: string
      description: Reference bases
      example: "G"
    alt:
      type: string
      description: Alternate (variant) vases
      example: "A"
```

CanDIG

Step 1: Endpoints

- Simple get/put (CR of CRUD) of each type of entity
- Try `python main.py` and look at http://localhost:8080/ui

```yaml
paths:
  /variants:
    get:
      operationId: main.get_variants
      summary: Get all variants within genomic range
      parameters:
        - name: chromosome
          in: query
          type: string
          pattern: "^[a-zA-Z0-9]*$"
        - name: start
          in: query
          type: integer
          minimum: 1
        - name: end
          in: query
          type: integer
      responses:
        200:
          description: Return variants
          schema:
            type: array
            items:
              $ref: '#/definitions/Variant'
    put:
      operationId: main.put_variant
      summary: Add a variant to the database
      parameters:
        - name: variant
          in: body
          schema:
            $ref: '#/definitions/Variant'
      responses:
        201:
          description: New variant created
        405:
          description: Cannot overwrite variant
```

Step 2: ORM

- Next step is to start building the underlying data representation
- Note: there are ways to get here from the Swagger representation, but we **don't** normally want that:
  - Underlying data model != API data structure
  - Swagger doesn't (can't) know about foreign keys, etc

```
class Individual(Base):
    """
    SQLAlchemy class/table representing an individual
    """
    __tablename__ = 'individuals'
    id = Column(Integer, primary_key=True)
    description = Column(String(100))
    created = Column(DateTime())


class Variant(Base):
    """
    SQLAlchemy class/table representing a Variant
    """
    __tablename__ = 'variants'
    id = Column(Integer, primary_key=True)
    chromosome = Column(String(10))
    start = Column(Integer)
    ref = Column(String(100))
    alt = Column(String(100))
    name = Column(String(100))
    created = Column(DateTime())


class Call(Base):
    """
    SQLAlchemy class/table representing Calls
    """
    __tablename__ = 'calls'
    id = Column(Integer, primary_key=True)
    individual_id = Column(String(20), ForeignKey('individuals.id'))
    variant_id = Column(String(20), ForeignKey('variants.id'))
    genotype = Column(String(20))
    fmt = Column(String(100))
    created = Column(DateTime())
```

orm.py

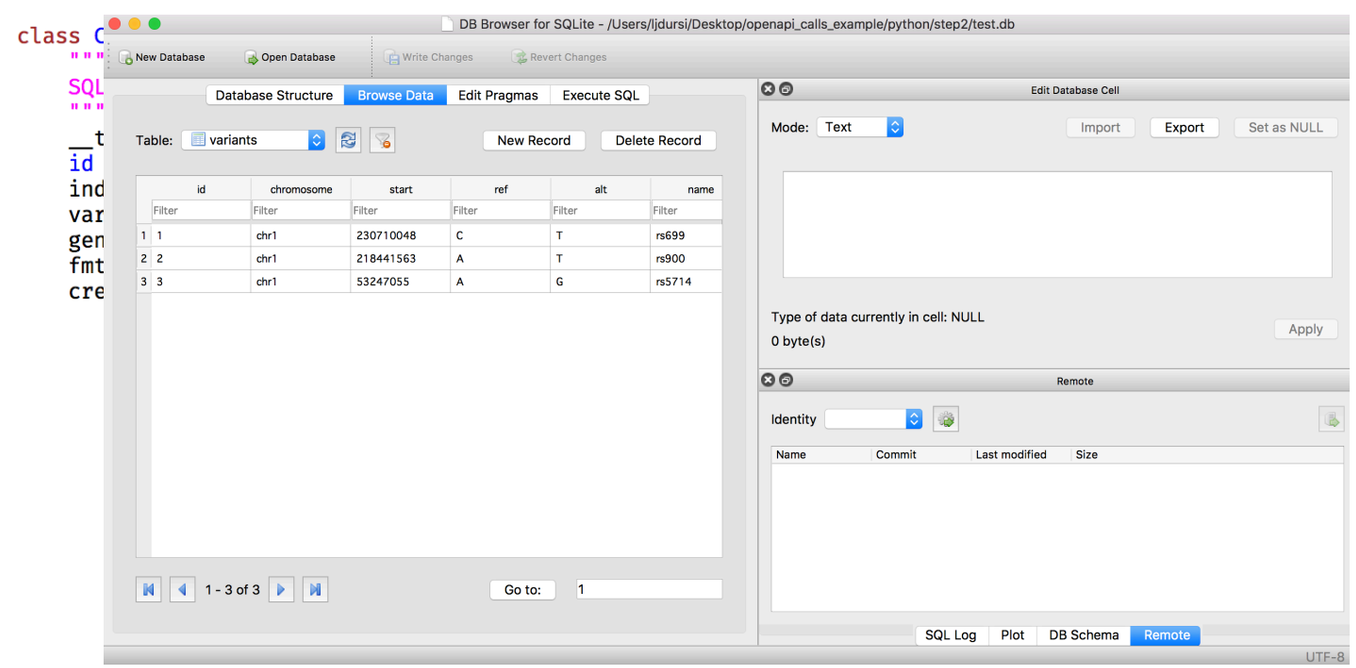
Step 2: ORM

- Next step is to start building the underlying data representation
- Note: there are ways to get here from the Swagger representation, but we **don't** normally want that:
  - Underlying data model != API data structure
  - Swagger doesn't (can't) know about foreign keys, etc
- python orm.py generates some test data:

orm.py

```
class Individual(Base):
    """
    SQLAlchemy class/table representing an individual
    """
    __tablename__ = 'individuals'
    id = Column(Integer, primary_key=True)
    description = Column(String(100))
    created = Column(DateTime())


class Variant(Base):
    """
    SQLAlchemy class/table representing a Variant
    """
    __tablename__ = 'variants'
    id = Column(Integer, primary_key=True)
    chromosome = Column(String(10))
    start = Column(Integer)
    ref = Column(String(100))
    alt = Column(String(100))
    name = Column(String(100))
    created = Column(DateTime())
```



Step 3: Implement Endpoints

- Once that's done, can start implementing the end points
- Get data from the DB, convert it into swagger-like data structures

```python
def get_variants(chromosome, start, end):
    """
    Return all variants between [chrom, start) and (chrom, end]
    """
    q = db_session.query(orm.Variant)
    q = q.filter_by(chromosome=chromosome).filter(and_(start ≥ start, start ≤ end))
    return [orm.dump(p) for p in q]


def get_individuals():
    """
    Return all individuals
    """
    q = db_session.query(orm.Individual)
    return [orm.dump(p) for p in q]


def get_calls():
    """
    Return all calls
    """
    q = db_session.query(orm.Call)
    return [orm.dump(p) for p in q]
```

step3/main.py

Step 4: More Complex Queries

- Finally, can start implementing more interesting queries
- Find all individuals associated with a variant, or variants associated with an individual
- Requires more interesting database operations, but the ORM makes this easy:

```yaml
/variants/by_individual/{individual_id}:
  get:
    operationId: main.get_variants_by_individual
    summary: Get variants called in an individual
    parameters:
      - $ref: '#/parameters/individual_id'
    responses:
      200:
        description: Return individuals
        schema:
          type: array
          items:
            $ref: '#/definitions/Variant'
      404:
        description: Individual does not exist

/individuals/by_variant/{variant_id}:
  get:
    operationId: main.get_individuals_by_variant
    summary: Get individuals with a given variant called
    parameters:
      - $ref: '#/parameters/variant_id'
    responses:
      200:
        description: Return individuals
        schema:
          type: array
          items:
            $ref: '#/definitions/Individual'
      404:
        description: Variant does not exist
```

step4/swagger.yaml

Step 4: More Complex Queries

- Finally, can start implementing more interesting queries
- Find all individuals associated with a variant, or variants associated with an individual
- Requires more interesting database operations, but the ORM makes this easy:
- 146 lines of python

```
def get_variants_by_individual(individual_id):
    """
    Return variants that have been called in an individual
    """
    ind_id = individual_id
    ind = db_session.query(orm.Individual).filter(orm.Individual.id == ind_id)
    if not ind:
        return NoContent, 404

    variants = [call.variant for call in ind.calls if call.variant is not None
    return [orm.dump(v) for v in variants], 200


def get_individuals_by_variant(variant_id):
    """
    Return variants that have been called in an individual
    """
    var_id = variant_id
    var = db_session.query(orm.Variant).filter(orm.Variant.id == var_id).one_o
    if not var:
        return NoContent, 404

    individuals = [call.individual for call in var.calls if call.individual is
    return [orm.dump(i) for i in individuals], 200
```

step4/main.py

Go and go-swagger

- Go-swagger takes a different (less pythonic, more go-ish) approach
- Generates template code directly, so can compile into small static binary

http://goswagger.io

Swagger 2.0 circleci passing | build passing | codecov 74% | slack 3/321

license Apache v2 | godoc reference | version 0.14.0 | container ready

Development of this toolkit is sponsored by VMware

vmware OPEN SOURCE

This package contains a golang implementation of Swagger 2.0 (aka OpenAPI 2.0): it knows how to serialize and deserialize swagger specifications.

Swagger is a simple yet powerful representation of your RESTful API.

> **Swagger in a nutshell** With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment.
>
> With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability. We created Swagger to help fulfill the promise of APIs.

Go and go-swagger

- Go-swagger takes a different (less pythonic, more go-ish) approach
- Generates template code directly, so can compile into small static binary
- Same steps - need a data layer (ORMish) and to implement the endpoints
- Similar number of lines of code (some things more automatic, some less)

```go
type ORMIndividual struct {
    ID uint `gorm:"primary_key,type:integer"`
    Description *string `gorm:"type:varchar(100)"`
    Created time.Time
}

func (o *ORMIndividual) from_Swagger (m models.Individual) {
    o.Description = m.Description
    o.Created = time.Time(m.Created)
    o.ID = uint(m.ID)
}

func (o ORMIndividual) to_Swagger() *models.Individual {
    m := new(models.Individual)
    m.Description = o.Description
    m.Created = strfmt.DateTime(o.Created)
    m.ID = int64(o.ID)
    return m
}

type ORMVariant struct {
    ID uint `gorm:"type:integer,primary_key"`
    Chromosome string `gorm:"type:varchar(10)"`
    Name *string `gorm:"type:varchar(100)"`
    Alt string `gorm:"type:varchar(100)"`
    Ref string `gorm:"type:varchar(100)"`
    Start uint `gorm:"type:integer"`
}
```

go/restapi/onfigure_variants_and_calls_api_demo.go