Pony ORM JavaScript Integration

Pony simplifies writing business logic at the backend. But what if you need to work with the same objects at the frontend? We thought it would be cool to have such an ability. Now you can do it using pony.js module.

The workflow consists of 5 steps:

1. Sending objects from the backend
2. Getting entity objects at the frontend
3. Working with objects at the frontend
4. Sending objects to the backend
5. Storing object in the database

Now let's look into the process in the detail.

1. Sending objects from the backend

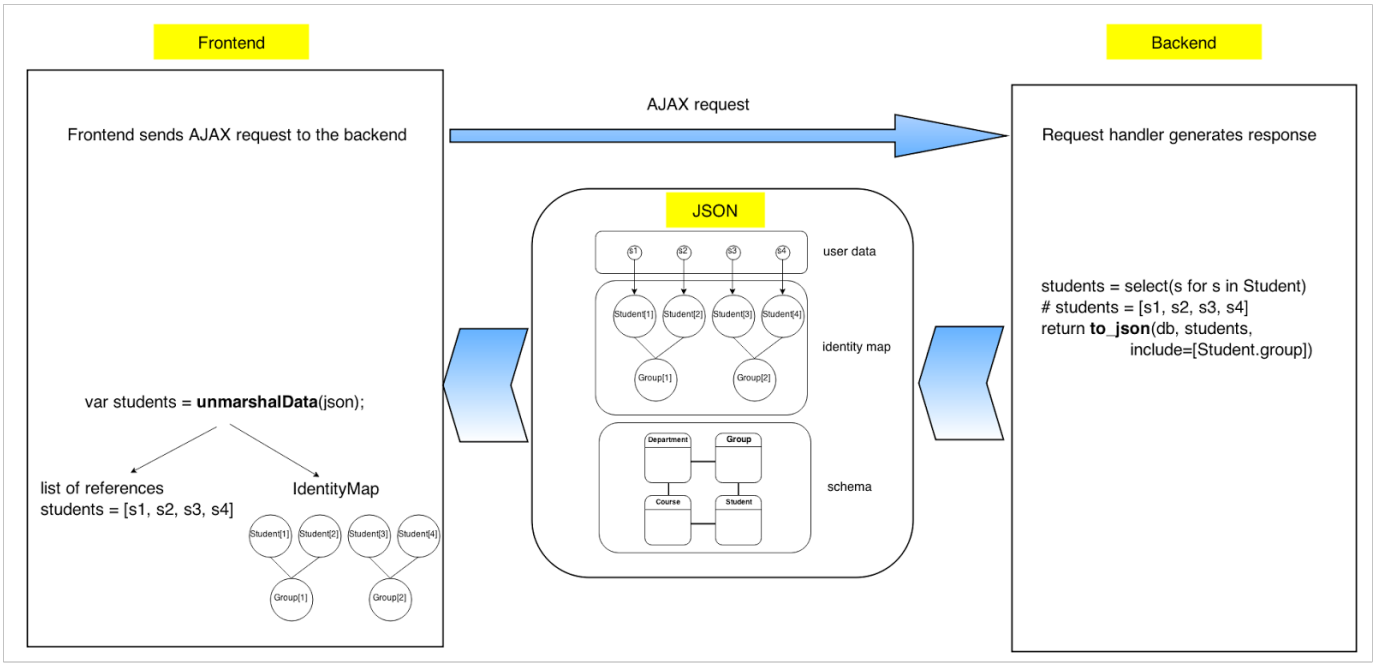
We start with extracting objects from the database using Pony queries:

```
students = select(s for s in Student).order_by(Student.name)
```

Now we are going to send them to the backend. For this purpose we use the function to_json():

```
to_json(db, students, include=[Student.group])
```

The **to_json()** function converts passed objects to JSON representation. By default, this function converts all attributes except lazy attributes and collections to-many. In case the object has to-one relationship, only the primary key of the related object will be included by default. If you need to get lazy attributes, collection or all attributes of the related to-one object, you need to specify this attribute in the **include** list.
The **to_json()** function returns a JSON structure which contains the database entity schema and all the necessary objects. The schema is needed in order to reconstruct entity objects with all relationships at the frontend.
Below is the diagram of the whole process:



2. Getting entity objects at the frontend

When the frontend received the JSON structure from the backend, it needs to unmarshal this data and link object the same way they are linked in the database. For this purpose you need to use the **unmarshalData()** function. As the result, the function returns the value which was passed to the **to_json()** function (list of students in our example) and also creates several the following dictionaries in the pony namespace:

pony.entities - contains entity definitions
pony.objects - contains entity objects, which can be referenced by a unique integer id (object's **_id_** attribute)

Each object has the **_pk_** attribute, which keeps the primary key of the object. In order to get an object by its primary key, you need to use the method **get_by_pk()** of an entity. For example:

```
var obj = pony.entities.Group.get_by_pk(103)
```

This method doesn't send any new requests to the backend. If the object exists in the database, but was not loaded to the frontend, it throws an exception, e.g. "Group with primary key 101 not found".

3. Working with objects at the frontend

At the moment, PonyJS is integrated with the KnockoutJS framework (http://knockoutjs.com/) and each entity object attribute is represented as an observable (http://knockoutjs.com/documentation/observables.html)
It allows easily establish two-way binding between entity objects and fronted UI using the MVVM model (http://knockoutjs.com/documentation/observables.html)
Each attribute is represented as a function. When you need to read the value of an attribute, you call this attribute without parameters. When you want to assign a new value, you pass this value as a parameter:

```
s = pony.entities.Student.get_by_pk(1)
var name = s. name()
s.name('New name')
```

Object keys

Each object has the following attributes with unique values:
Instance._pk_ - database's primary key. Equal to null for a newly created object.
Instance._id_ - unique integer id, assigned from a sequence at the frontend

Creating new objects

In order to create a new object, you need to call the **create()** method of entity:

```
pony.entities.Student.create({name: 'John'})
```

attrs is an optional parameter that allows to specify the attribute values.
You also can call the **create()** method of a relationship attributes. In this case a newly created object will automatically have a relationship:

```
g = pony.entities.Group.get_by_pk(1)
g.students.create({name: 'John'})
```

Establishing and dropping relationship

Each relationship attribute to-many has **add()** and **remove()** methods:

```
s1 = pony.entities.Student.get_by_pk(1)
g2 = pony.entities.Group.get_by_pk(2)
g2.students.add(s1)
```

Deleting an object

Method destroy() deletes an object:

```
s = pony.entities.Student.get_by_pk(1)
s.destroy()
```

Data modification flag

pony.cache_modified is an observable that will be set to true once any entity object is modified.

toString method

You can use object's **toString** method for debug purposes.

4. Sending objects to the backend

When you want to store your updated objects in the database, you use the function **pony.getChanges(**additional_info**).** It returns a JSON structure that needs to be sent to the backend.
'additional_info' can be any JSON data or Pony entity objects.

5. Storing object in the database

JSON structure that is received from the frontend should to be passed to the save_changes() function:

```
additional_info = save_changes(db, json):
```

Don't forget to decorate the function where you call this method with the @db_session decorator.