Implementing API Exceptions

It's very common to implement RESTful APIs on top of Flask. One of the first things that developers run into is the realization that the builtin exceptions are not expressive enough for APIs and that the content type of *text/html* they are emitting is not very useful for API consumers.

The better solution than using `abort` to signal errors for invalid API usage is to implement your own exception type and install an error handler for it that produces the errors in the format the user is expecting.

Simple Exception Class

The basic idea is to introduce a new exception that can take a proper human readable message, a status code for the error and some optional payload to give more context for the error.

This is a simple example:

```
from flask import jsonify

class InvalidUsage(Exception):
    status_code = 400

    def __init__(self, message, status_code=None, payload=None):
        Exception.__init__(self)
        self.message = message
        if status_code is not None:
            self.status_code = status_code
        self.payload = payload

    def to_dict(self):
        rv = dict(self.payload or ())
        rv['message'] = self.message
        return rv
```

A view can now raise that exception with an error message. Additionally some extra payload can be provided as a dictionary through the *payload* parameter.

Registering an Error Handler

At that point views can raise that error, but it would immediately result in an internal server error. The reason for this is that there is no handler registered for this error class.

That however is easy to add:

```python
@app.errorhandler(InvalidUsage)
def handle_invalid_usage(error):
    response = jsonify(error.to_dict())
    response.status_code = error.status_code
    return response
```

Usage in Views

Here is how a view can use that functionality:

```python
@app.route('/foo')
def get_foo():
    raise InvalidUsage('This view is gone', status_code=410)
```