coursebrew

Back End Developer's Guide

coursebrew's back end is created using Flask. The main process flow of a Flask function can be generalized to:

1. A URL request sends a payload to the server and is mapped to a function
2. Python runs some code to handle the payload
3. A return is sent back to the origin of the request with a modified payload

So let's take a step through a Flask function, which adds an instructor.
So from our front end Angular code, we had this function to add an instructor.

```
addInstructor(instructor: Instructor): Observable<any>{
      return this.http
         .post(this.appGlobal.baseAppUrl +
this.appGlobal.basePort + "/add_instructor",
JSON.stringify(instructor))
}
```

The bolded code sends a request to the IP of our Flask server with an appended path, which is "/add_instructor".

This path correlates to a function in our **handlebars.py** file.

```
@bp.route('/add_course', methods=['POST', 'GET'])
def bp_add_course():
  return courses.add_course()
```

`@bp.route` binds the path to a function, so whenever a request is sent to the path, our function will run. Methods set the rules for what kind of requests the client is allowed to make, which is POST and GET as you can see above. We've separated the actual code for our function into another file to make the system more manageable. That code is located in **courses.py**

```
def add_course():
   try:
      data_j = json.loads(request.data)
      course_id = data_j['course_id']
      name = data_j['name']
      owner = data_j['owner']
      year = data_j['year']
      freq_spr = data_j['freq_spr']
      freq_sum1 = data_j['freq_sum1']
      freq_sum2 = data_j['freq_sum2']
      freq_fal = data_j['freq_fal']
      freq_spr_l = data_j['freq_spr_l']
      freq_sum1_l = data_j['freq_sum1_l']
      freq_sum2_l = data_j['freq_sum2_l']
      freq_fal_l = data_j['freq_fal_l']
      token = data_j['token']
      db = get_db()
      db_courses = db.execute(
         'SELECT course_id'
         ' FROM courses'
      )
      desc = db_courses.description
      column_names = [col[0] for col in desc]
      data= [dict(zip(column_names, row))
         for row in db_courses.fetchall()]
      for i in data:
         if(course_id == i['course_id']):
            db.close()
            return jsonify({'message': 'duplicate course, course
not added'}), 200
      db.execute(
         'INSERT INTO courses (course_id, name, owner, year,
freq_spr, freq_sum1, freq_sum2, freq_fal, freq_spr_l,
freq_sum1_l, freq_sum2_l, freq_fal_l, owner, token)'
         ' VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',
         (course_id, name, owner, year, freq_spr, freq_sum1,
freq_sum2, freq_fal, freq_spr_l, freq_sum1_l, freq_sum2_l,
freq_fal_l, owner, token)
         )
```

```
        db.commit()
        create_sections(db, data_j, token)
        db.close()
        admin.create_log(request, course_id + " added to courses")
        return jsonify({'message': 'course added'}), 200
    except:
        admin.create_log(request, "Error caught in add_course()
located in courses.py")
        return jsonify({'message': 'error encountered, please
contact server administrator'})
```

1. "`request.data`" holds the payload that was sent from our client.
2. "`get_db`" gets us our database object
3. "`db.execute`" takes a SQL syntax string and runs whatever SQL command you put in
4. "`db.commit`" saves the db changes
5. `return jsonify` returns a message to our client.

The main takeaway is, if you want to add a backend function, add a path to the handlebars.py file and connect it to the code you want to run.