OBJECTIVES

- Understand how to translate postgres fields to JSON API format
- Understand how to camelize and decamelize object properties
- Know how to use Boom node module to throw errors

Which of these two JSON formats is correct?

Format 1

```
{
   "id": 1,
   "name": "Bootsy",
   "age": 3,
   "skill": "birding",
   "description": "Calico Cat",
   "profile_url": "meow.png",
   "created_at": "2016-11-15T16:00:55.104Z",
   "updated_at": "2016-11-15T16:00:55.104Z"
}
```

Format 2

```
{
    "id": 1,
    "name": "Bootsy",
    "age": 3,
    "skill": "birding",
    "description": "Calico Cat",
    "profileUrl": "meow.png",
    "createdAt": "2016-11-15T16:00:55.104Z",
    "updatedAt": "2016-11-15T16:00:55.104Z"
}
```

FROM THE GOOGLE JSON STYLE GUIDE PROPERTY NAMES SHOULD...

- Be meaningful names with defined semantics.
- Must be camel-cased, ascii strings.
- The first character must be a letter, an underscore (_) or a dollar sign ($).
- Subsequent characters can be a letter, a digit, an underscore, or a dollar sign.
- Reserved JavaScript keywords should be avoided.

What do you do if you used an underscore naming convention for your database fields?

```
When we send things out from our database we need to translate:
profile_url => profileUrl, created_at => createdAt, etc.

When we recieve requests we need to go the other way:
profileUrl => profile_url
```

CAMELIZEKEYS AND DECAMELIZEKEYS TO THE RESCUE!

```
const { camelizeKeys, decamelizeKeys } = require('humps');

camelizeKeys({hello_world: 'howdy'})   // {helloWorld: 'howdy'}
decamelizeKeys({theCats: 'meow'})      // {the_cats: 'meow'}
```

THROWING ERRORS WITH NODE

```
router.post('/', (req, res, next) => {

  const { name, skills, description, profileUrl } = req.body;

  if (!name || !name.trim()) {
    res.status(400).send("Name must not be blank");
    return;
  }

  // more cool code ...
}
```

THROWING ERRORS WITH BOOM

```
const boom = require('boom')

router.post('/', (req, res, next) => {

  const { name, skills, description, profileUrl } = req.body;

  if (!name || !name.trim()) {
    next(boom.create(400, 'Name must not be blank.'));
    return;
  }

  // more cool code ...
}
```

WHAT ARE SOME ADVANTAGES TO THE BOOM APPROACH?

- centralized error logging
- ability to handle errors differently in development vs. production
- not "rollling your own" in your routers (which is error prone)

REFERENCES:

https://github.com/domchristie/humps

https://google.github.io/styleguide/jsoncstyleguide.xml#PropertyNameFormat

https://github.com/hapijs/boom