REST API

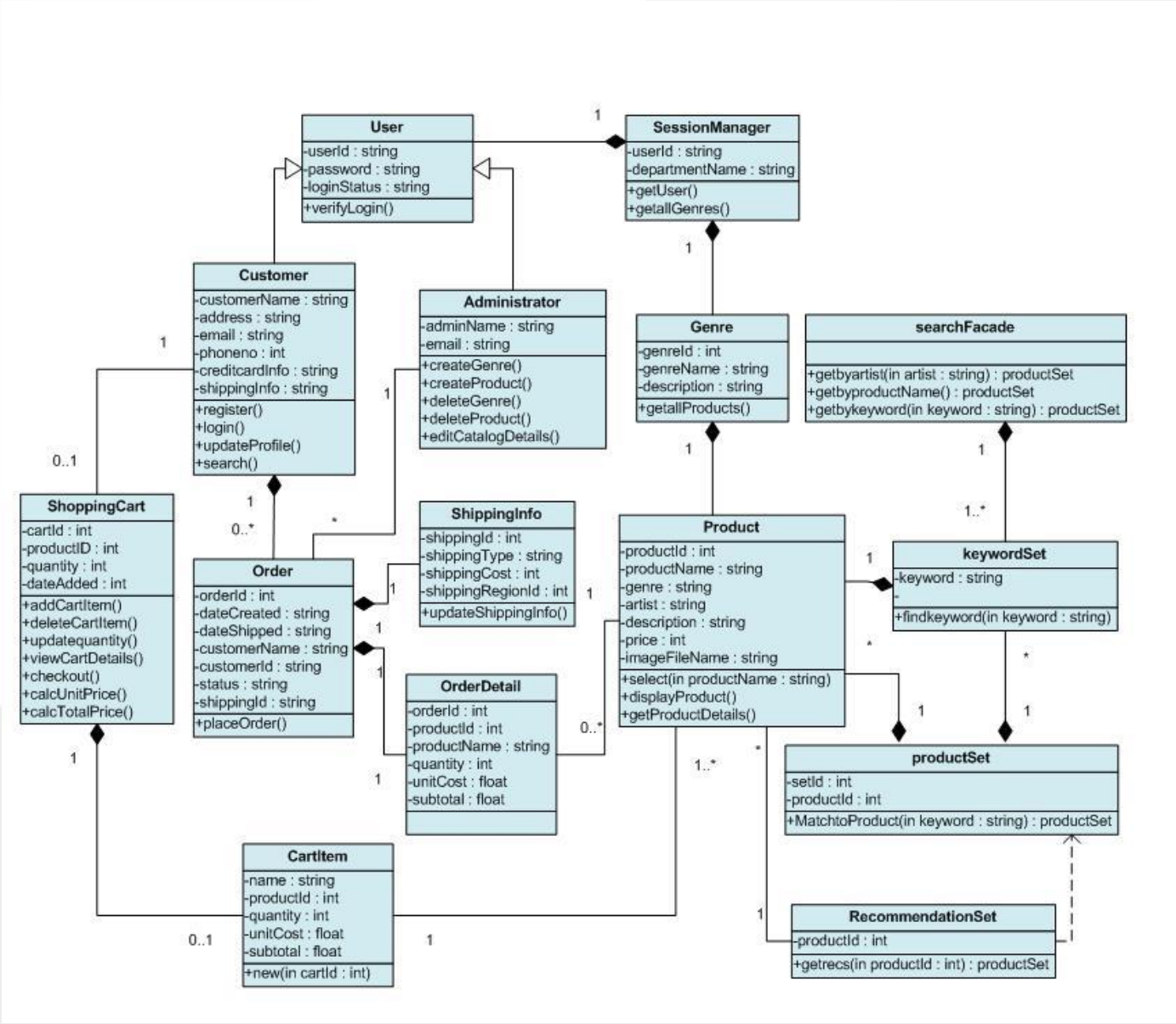
- **RE**presentational **S**tate **T**ransfer: a set of rules that developers follow when they create their API
- REST API: Application Programming Interface that follows REST principles
- It is a software architecture style for building scalable web services
- REST is independent of any underlying protocol and is not necessarily tied to HTTP
- stateless: no client context being stored on the server between requests

Application Programming Interface (recap)

- a set of clearly defined methods of communication between various software components
- `toUpperCase()` is an API for converting a lowercase letter to uppercase
- `'a' ^ 0x20` is also a way for converting a lowercase letter to uppercase

Use Case

- What kind of resource should be returned?
- What protocol?
- What kind of data format?



Terminology

- resource: a piece of information
- URI: Uniform Resource Identifier
- Representation: the way to represent a resource
- Uniform interface: a way to interact with resources

Resource

- a resource is a piece of information
- a resource can be anything: a document, a photo, a collection of other resources, etc.
- can be stored in the server and represented as a stream of bits
- similar to an object in OOP
- identified by a URI

URI

- Uniform Resource Identifier
- a string of characters that identifies a resource
- a URI can be a URL or a URN
- be descriptive (human readable) and consistent
  - http://example.com/articles/1
  - http://example.com/map/roads/USA/CA/17_mile_drive
  - http://example.com/courses/CS229/code.tar.gz
  - http://example.com/relationships/animal/dog

Representation

- a resource can be represented in different ways
- but representation must refer to a unique resource
- 'noun' vs 'verb': use nouns instead of verbs
  - http://example.com/articles/1
  - http://example.com/articles/1/delete
- need not have structure but are valuable for the clients to navigate the API

Uniform Interface

- resourced-based: resources are identified by URIs
- manipulation of resources through representations
- self-descriptive messages
- hypermedia as the engine of application state (HATEOAS)
  - clients deliver state via body, query-string parameters, request headers and the requested URI
  - servers deliver state via body, response codes, and response headers
  - links are used to navigate between resources

REST API Example

user-route.js

method	request params	request body	response body
GET	empty	empty	list of users
GET	/:id	empty	user
POST	empty	user	user
PUT	/:id	user	user
DELETE	/:id	empty	empty

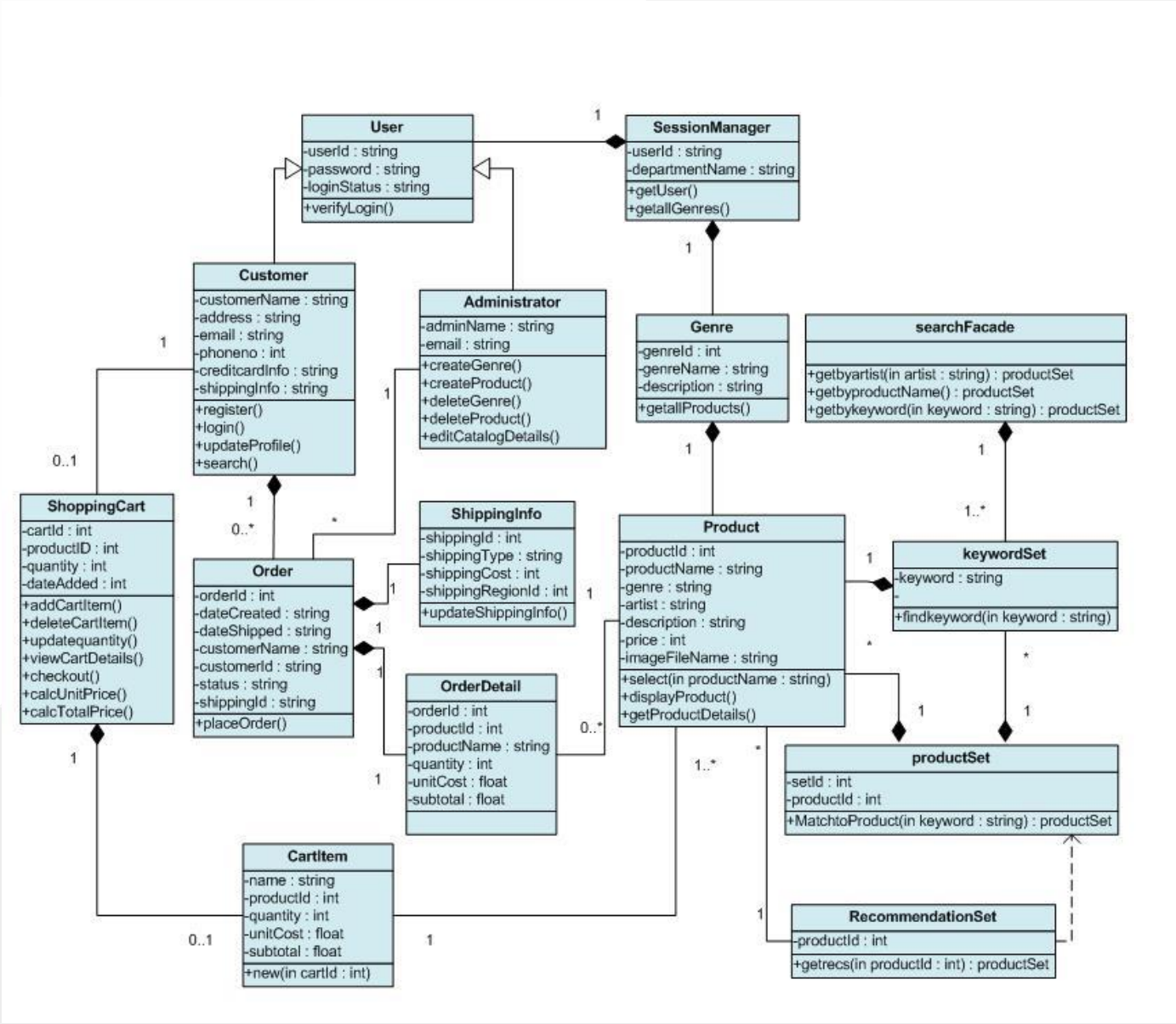
Http Status Codes

- 1xx: informational
- 2xx: success
  - 200: OK
  - 201: created
  - 202: accepted
- 3xx: redirection
  - 301: moved permanently
  - 302: found
  - 304: not modified
- 4xx: client error
  - 400: bad request
  - 401: unauthorized
  - 403: forbidden
  - 404: not found
- 5xx: server error
- create your own status codes under 5xx

REST API Design

1. Figure out the data set
2. Break it into resources
3. Design the URIs
   a. name the resources with URIs
   b. expose a subset of uniform interface (CRUD)
   c. design data in JSON
   d. configure representation accepted from clients and served to clients
   e. error handling with status codes

Use Case (cont'd)



Disadvantages of REST

- no standard
- no official documentation
- not all resources fit into the REST model - login, logout, etc.
- status codes need to be unified
- 404 - interface is not discoverable or resource is not available?

Suggestions when designing REST API

- use plural nouns instead of verbs
- use HTTP verbs, but no need to use all of them
- use sub-resources for relations
- 100% RESTful API? Impossible!

API Testing

- Postman
- curl / httpie