# New Yorker Exercise

```
> cat whoami.js
```

```
import { Person } from 'humanity'
```

```
const countries = Object.freeze({'Austria': 'Austria'} )
```

```
const phil = new Person('Philipp', 34, countries.Austria)
phil.studies = ['IT Sec', 'Software Engineering']
```

```
function* work(person) {
    console.log('Curriculum vitae')

    while(person.age <= 67) {
        let {title, company} = yield `${person.name} work`
        console.log(`${company}, ${title}`)
    }
}

const career = work(phil)
career.next()


career.next({title: 'Dev & Pentester', company: 'BaseCamp'})


career.next({title: 'Senior Software Engineer',
             company: 'RadarServices'})
```

# Projects



Crawler
Data Pipeline
Pentest automation
Security Scanner
DTE
SOC Automation

Time Tracker
Report Gen

More please!
:)

# How to add Indian Standard Time in Django?

117

Change the field `TIME_ZONE` in the `settings.py`. For the Indian standard time you will need:

```
TIME_ZONE =  'Asia/Kolkata'
```

For more information about the TIME_ZONE in Django you can see: https://docs.djangoproject.com/en/dev/ref/settings/#time-zone

share edit delete flag

answered Feb 18 '14 at 13:44

Jon

5,679 ● 3 ● 31 ● 64

## The task

- Parse a Nginx access.log
- Expose data through GraphQL
- Authentication (if time)
- Dockerize (if time)

## Technologies

- Flask, Postgres, Docker(obligatory)
- **DB**: SQLAlchemy, psycopg2
- **Auth**: Bcrypt, JWT_extended
- **GraphQL**: Graphene, Flask-GraphQL
- **Deploy**: gunicorn, supervisor

- **CLI**: click
- **Tests**: pytest, coverage, factory
- **Type hints**: Python typing
- **Linting**: flake8

# Basic flow

- Test user is created at built time

```
flask adduser tester tester123456 tester@nydata.com
```

- Get token `/auth/get_token/'`

  POST with username and password → `'access_token'`

- Query API `/graphql/?query={query}'`

  GET with query &
  `headers={'Authorization': f'Bearer {token}'`

# Query

```
query = 'query{logs(dateFrom:"2000-01-01"' \
        ',dateTo:"2019-12-12") ' \
        '{hostIP verb userAgent timestamp}}'

url = f'http://localhost:5000/graphql/?query={query}'
headers = {'Authorization': f'Bearer {token}'}

graphql_response = requests.get(url, headers=headers)
```

```
HTTP/1.0 200 OK
Content-Length: 2289
Content-Type: application/json
Date: Mon, 09 Dec 2019 20:15:43 GMT
Server: Werkzeug/0.16.0 Python/3.7.4

{
    "data": {
        "logs": [
            ...
            {
                "hostIP": "77.179.66.156",
                "timestamp": 1481103803.0,
                "userAgent": "Mozilla/5.0 ...",
                "verb": "GET"
            },
            ...
```

**Architecture**

- app

- user, auth & log_parser

- general: database, utils, compat

- cli commands

Everything uses blueprints & app factory pattern

# Do you want to see the real code?

https://github.com/gardiac2002/nydata_sample

**app.js**

- create_app (app factory pattern)

- register_extensions
- register_blueprints
- register_shellcontext (flask shell)
- register_commands (cli)

**.env → Environment variable in docker**

```
FLASK_APP=autoapp.py
```

**autoapp.py**

```
from nydata.app import create_app
app = create_app()
```

**User**

- User & role model

- Passwords

- add_user (used by cli)

What to improve?

- Password criterias (length etc.)

- Functionality to level up algorithm

```
class User(UserMixin, SurrogatePK, Model):

username = Column(db.String(80), unique=True,nullable=False)
email = Column(db.String(80), unique=True, nullable=False)

# Refactoring: I would not allow null passwords in future.
password = Column(db.LargeBinary(128), nullable=True)
...

def __init__(self, username, email, password=None,**kwargs):
    ...
    if password:
        self.set_password(password)
    ...
def set_password(self, password):
  self.password=bcrypt.generate_password_hash(password)

def check_password(self, value):
  return bcrypt.check_password_hash(self.password,
                                    value)
```

**Auth**

- authentication through `/get_token/`

What to improve?

- API token management (refresh…)

- An index on User.username

```
blueprint = Blueprint("auth", __name__, ...)

@blueprint.route("/get_token/", methods=["POST"])
def get_token():


... a lot of checks!

user = User.query.filter_by(username=username).first()

if user is not None and user.check_password(password):
  access_token = create_access_token(identity=username)
  return jsonify(access_token=access_token), 200

return jsonify({"msg": ""}), HTTPStatus.UNAUTHORIZED
```

**log_parser**

- parsing of log files

- saving to models

What to improve?

- run regular worker (e.g. celery)

- move to own library

- collision check with hashlib

# Idea of log_parser

```
raw_string
      |> pattern
      |> transformer
      |> parser

       → LogEntry

pattern & transformer registered
      → NGINX = "nginx"
```

# Nginx pattern

```
_NGINX_LOG_LINE = r"""^(?P<ip>[0-9.]*)     # ip entry
                    \s*-\s*-\s*
                    \[(?P<time>.*?)\]    # datetime
                    \s"
                    (?P<verb>[A-Z]*)     # HTTP method
                    \s
                    (?P<path>.*?)
                    \sHTTP/\d\.\d"\s
                    (?P<code>\d{3})      # HTTP status
                    \s\d*\s".*?"\s
                    "(?P<agent>.*?)"     # agent
                    $"""
```

# transformer

```
def transform_nginx(log_entry: Dict) -> LogEntry:

  def get(name: str, default=""):
      return log_entry.get(name, default)

  original_date_time = parser.parse(get("time"), fuzzy=True)
  timestamp = int(original_date_time.timestamp())
  code = int(get("code"))

  return LogEntry(host_ip=get("ip"),
        original_date_time=original_date_time,
        timestamp=timestamp,
        request_verb=get("verb"),
        request_path=get("path"),
        response_code=code,
        user_agent=get("agent"),
    )
```

# Parser

```
def generic_parse_logs(...) -> List[LogEntry]:
    ...choose pattern and transformer by key
    parse_line = partial(extract_and_transform,
                         pattern, transform)

    return [parse_line(line) for line in
            log_file.splitlines()]


def extract_and_transform(pattern, transform, line):
    matched = pattern.match(line)
    if matched is None:
        return None

    extracted = matched.groupdict()
    return transform(extracted)
```

# log_parser interface

**Data**

- LogEntry → dataclass
- NGINX

**Generic**

```
- digest_log_by_line(source: str, file_path: str)

#                nginx, apache
- digest_logs(source: str, file_path: str)
```

**Specific**

```
- parse_and_save_nginx_logs(file_path: str,
                            parse_by_line_at_mb: float)
```

# log_parser :: Data

```
NGINX = "nginx"

@dataclass(frozen=True, eq=True)
class LogEntry:

    host_ip: str
    original_date_time: datetime
    timestamp: int

    request_verb: str  # GET | POST etc
    request_path: str

    response_code: int
    user_agent: str
```

```
def parse_and_save_nginx_logs(file_path: str, byline_at_mb:
                              float) -> bool:

    try:
        size_in_mb = path.getsize(file_path) / 1_000_000.0
        bigger_than_threshold = size_in_mb > byline_at_mb

        if bigger_than_threshold:
            return digest_log_by_line(NGINX, file_path)

        else:
            return digest_logs(NGINX, file_path)

    except Exception as error:
        eprint(f"[-] Failed ... location={file_path}")
        eprint(f"[-] Retrieved error: {error}")
        return False
```

# digest_log_by_line

```
def digest_log_by_line(source: str, file_path: str) -> bool:

    num_errors = 0
    failed_lines = []

    with open(file_path) as file_conn:
        for line in file_conn:
            try:
                log_entry = parse_log_line(source, line)
                save_log_to_database(source, log_entry)

            except Exception as error:
                ... error tracking
                continue
        ...
```

# log_parser :: db models

```
class LogSource(SurrogatePK, Model):
    name = Column(db.String(80), unique=True)
    lines = relationship("LogLine",
                         backref="log_source")


class LogLine(SurrogatePK, Model):
    ...
    hash_code = Column(db.String(128), index=True)
    log_source_id = Column(db.Integer(),
                           db.ForeignKey("log_sources.id"))
```

# log_parser :: schema.py

```
class Log(SQLAlchemyObjectType):

    class Meta:
        model = LogLine
        interfaces = (graphene.relay.Node,)


class Query(graphene.ObjectType):

    logs = graphene.List(
        Log,
        date_from=graphene.Date(required=True),
        date_to=graphene.Date(required=True),
    )
    def resolve_logs(self, info, date_from, date_to):
        ... filter by date from → to
```

# log_parser :: view

```
def graphql_view():
    view = GraphQLView.as_view("", schema=schema,
                                   graphiql=False)
    return jwt_required(view)


blueprint = Blueprint("graphql", ...)
blueprint.add_url_rule("/", view_func=graphql_view())
```

# Tests

52 unittests

12 Fixtures

1 Factory

Coverage 89%

```
---------- coverage: platform linux, python 3.7.5-final-0 -
Name                                          Stmts   Miss  Cover
-----------------------------------------------------------------
nydata/__init__.py                                0      0   100%
nydata/app.py                                    60     14    77%
nydata/auth/__init__.py                           1      0   100%
nydata/auth/views.py                             20      1    95%
nydata/commands.py                               48     24    50%
nydata/compat.py                                 13      5    62%
nydata/database.py                               44      6    86%
nydata/error.py                                  12      0   100%
nydata/extensions.py                             16      0   100%
nydata/log_parser/__init__.py                     3      0   100%
nydata/log_parser/data.py                        11      0   100%
nydata/log_parser/log.py                         46      0   100%
nydata/log_parser/models.py                      66      4    94%
nydata/log_parser/parser/__init__.py              1      0   100%
nydata/log_parser/parser/parser.py               32      0   100%
nydata/log_parser/parser/pattern.py               9      0   100%
nydata/log_parser/parser/transform.py            13      0   100%
nydata/log_parser/schema.py                      20      0   100%
nydata/log_parser/views.py                       14      0   100%
nydata/settings.py                               27      0   100%
nydata/user/__init__.py                           2      0   100%
nydata/user/models.py                            58      4    93%
nydata/user/views.py                              3      0   100%
nydata/utils.py                                  14      0   100%
-----------------------------------------------------------------
TOTAL                                           533     58    89%
```

## Missing from presentation

- Command line tools
- Configuration
- General tools e.g database & compat

Thank you for your wonderful attention! :-)