Advanced Software Engineering (LAB)

Stefano Forti

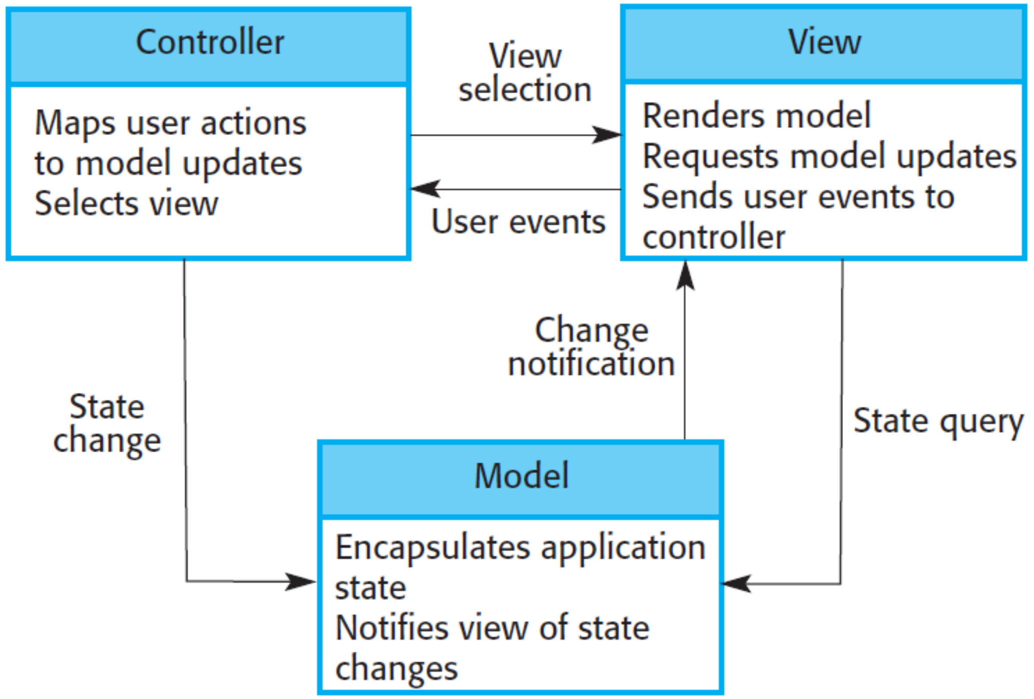
`name.surname@di.unipi.it`

Department of Computer Science, University of Pisa

Q
&
A

Previously on ASE…

Model-View-Controller

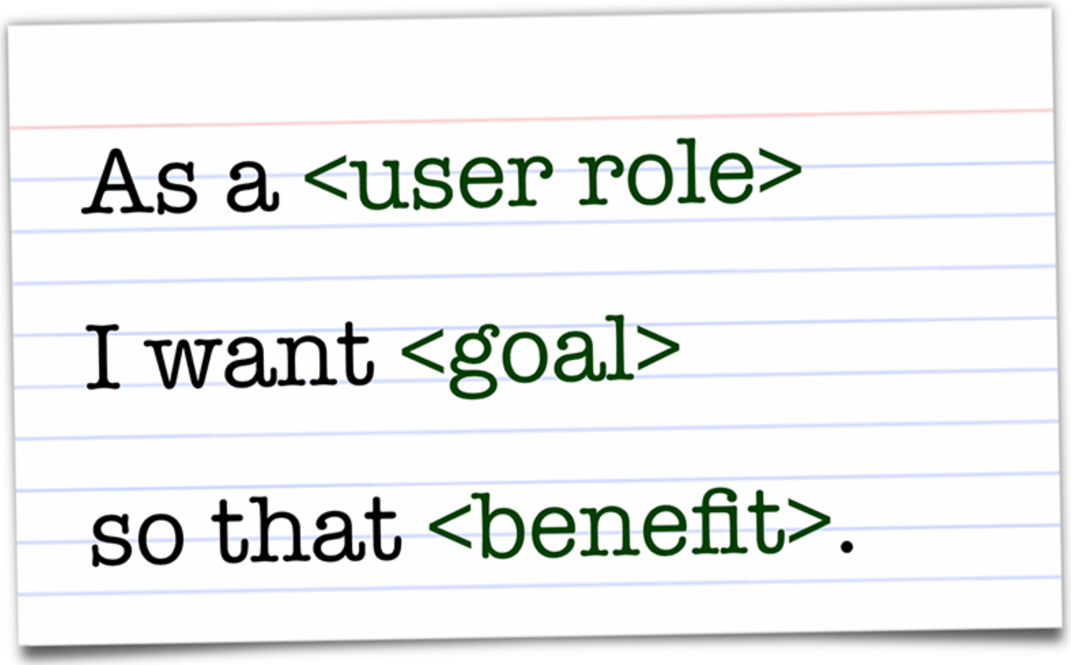


- **Model** – manages the data
- **View** – displays the Model for a particular context (e.g. web view, PDF)
- **Controller** – manipulates the Model to change its state

Beep Beep

Beep Beep offers a web view where users can see their runs, races, and training plans, all in one glimpse.

User Stories



As a <user role>

I want <goal>

so that <benefit>.

- Simple descriptions of interactions users have with an application, usually written when a project starts.

User Stories

As a	Unregistered User
I want to	Register
So that	I can use app functionalities
Priority	1.1

As a	Registered User
I want to	Authenticate
So that	I can hook to Strava & access my data
Priority	1.2

As a	Connected User
I want to	Logout
So that	I can disconnect myself
Priority	1.2

As a	Connected User
I want to	See all my previous runs on a list
So that	I can keep track of them
Priority	1.3

As a	Connected User
I want to	See average speed of all my runs
So that	I can improve myself
Priority	1.4

As a	Registered User
I want to	Delete my account
So that	All my data will be deleted
Priority	1.5

As a	Connected User
I want to	Click on a previous run
So that	To see speed & distance of a single run
Priority	2.1

As a	Connected User
I want to	Set a training objective
So that	I can plan my running
Priority	2.1

As a	Connected User
I want to	Challenge a previous run
So that	I can challenge myself
Priority	2.2

As a	Connected User
I want to	Compare different run statistics
So that	I can improve myself
Priority	2.3

As a	Connected User
I want to	See distance to my set training objective
So that	I know how much to run
Priority	2.4

As a Registered User
I want to Get a configurably periodic report via email
So that I can get them when I feel so
Priority 3.1

As a Registered User
I want to Get tips on when to run (based on my current training objective and weather forecast)
So that I can run in my ideal conditions
Priority 3.2

App Components

- We give you a skeleton app implementing these stories:

As a	Unregistered User
I want to	Register
So that	I can use app functionalities
Priority	1.1

As a	Registered User
I want to	Authenticate
So that	I can hook to Strava & access my data
Priority	1.2

As a	Connected User
I want to	Logout
So that	I can disconnect myself
Priority	1.2

As a	Connected User
I want to	See all my previous runs on a list
So that	I can keep track of them
Priority	1.3

Checklist

- A Linux distro properly installed (e.g., Ubuntu, lubuntu)
- Python 3.6+ and Flask.
- Redis and Celery:

```
pip install redis
```

```
pip install celery
```

Future work

- Fork the primer code at: https://github.com/ase-unipi/BeepBeepPrimer
- Today: implement all High Priority stories.
- At home: implement all Medium Priority stories
- Bonus: implement at least one Low Priority story.

As a	Unregistered User
I want to	Register
So that	I can use app functionalities
Priority	1.1

As a	Registered User
I want to	Authenticate
So that	I can hook to Strava & access my data
Priority	1.2

As a	Connected User
I want to	Logout
So that	I can disconnect myself
Priority	1.2

As a	Connected User
I want to	See all my previous runs on a list
So that	I can keep track of them
Priority	1.3

As a	Connected User
I want to	See average speed of all my runs
So that	I can improve myself
Priority	1.4

As a	Registered User
I want to	Delete my account
So that	All my data will be deleted
Priority	1.5

As a	Connected User
I want to	Click on a previous run
So that	To see speed & distance of a single run
Priority	2.1

As a	Connected User
I want to	Set a training objective
So that	I can plan my running
Priority	2.1

As a	Connected User
I want to	Challenge a previous run
So that	I can challenge myself
Priority	2.2

As a	Connected User
I want to	Compare different run statistics
So that	I can improve myself
Priority	2.3

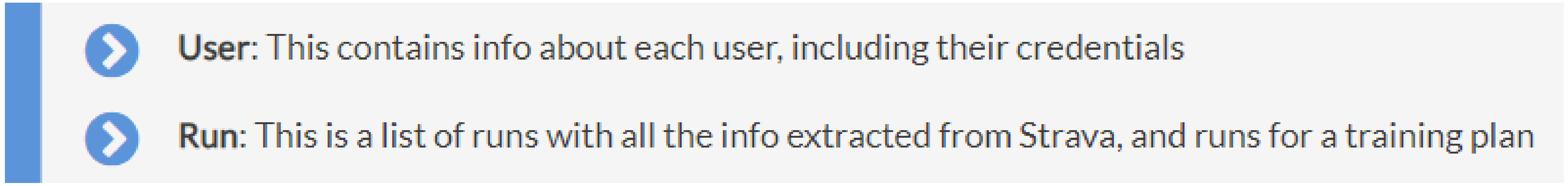
As a	Connected User
I want to	See distance to my set training objective
So that	I know how much to run
Priority	2.4

As a	Registered User
I want to	Get a configurably periodic report via email
So that	I can get them when I feel so
Priority	3.1

As a	Registered User
I want to	Get tips on when to run (based on my current training objective and weather forecast)
So that	I can run in my ideal conditions
Priority	3.2

Skeleton Model

- 2 database tables



- **User**: This contains info about each user, including their credentials
- **Run**: This is a list of runs with all the info extracted from Strava, and runs for a training plan

implemented using **Flask-SQLAlchemy**.

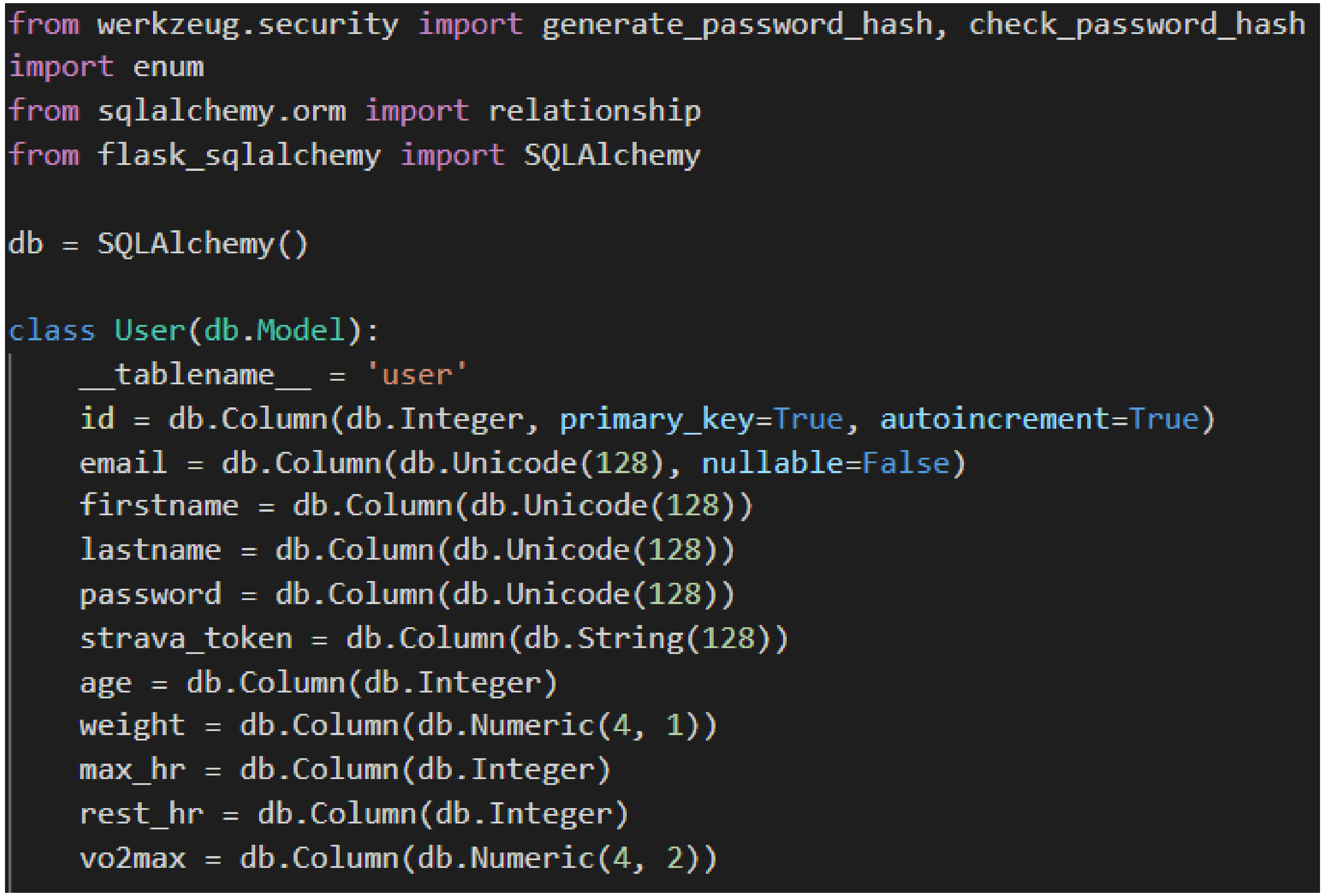
- If you need more, you can add other tables.

SQLAlchemy

- SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.
- Used at

Flask-SQLAlchemy

http://flask-sqlalchemy.pocoo.org/2.3/

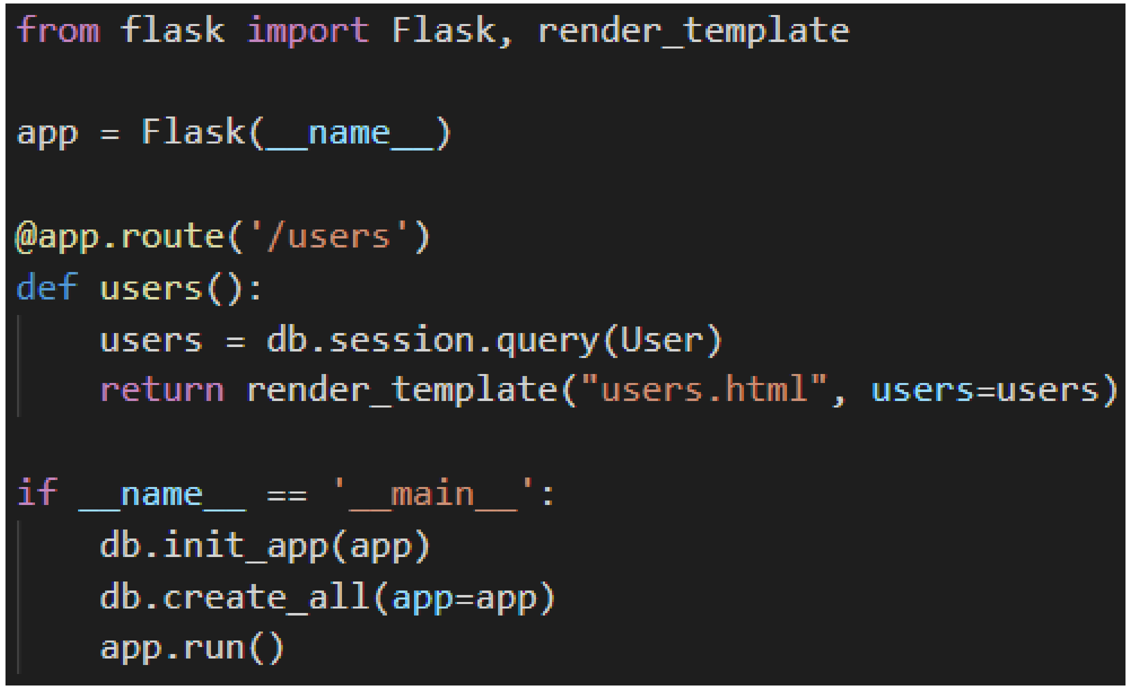


```python
from werkzeug.security import generate_password_hash, check_password_hash
import enum
from sqlalchemy.orm import relationship
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class User(db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    email = db.Column(db.Unicode(128), nullable=False)
    firstname = db.Column(db.Unicode(128))
    lastname = db.Column(db.Unicode(128))
    password = db.Column(db.Unicode(128))
    strava_token = db.Column(db.String(128))
    age = db.Column(db.Integer)
    weight = db.Column(db.Numeric(4, 1))
    max_hr = db.Column(db.Integer)
    rest_hr = db.Column(db.Integer)
    vo2max = db.Column(db.Numeric(4, 2))
```

- You can specify the tables using `Model` as base class.
- Flask-SQLAlchemy wraps all calls to SQLAlchemy and exposes a session object to your Flask app views to manipulate the model.

Skeleton View

- When a request is received, and a view is invoked, SQLAlchemy sets up a DB session object inside an application context.
- We use **Jinja** functions (embedded in Flask) to “compose” the view.

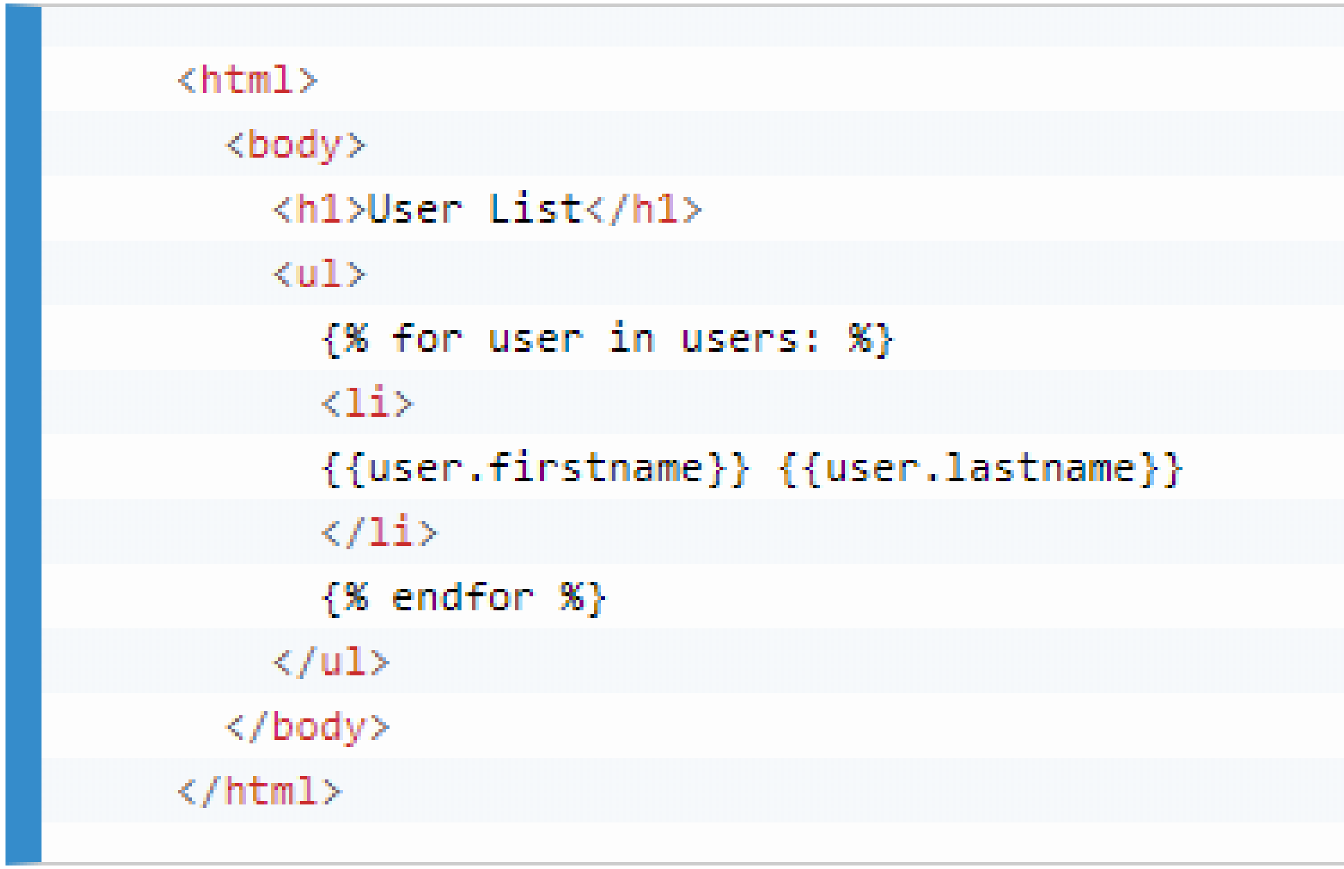


```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/users')
def users():
    users = db.session.query(User)
    return render_template("users.html", users=users)

if __name__ == '__main__':
    db.init_app(app)
    db.create_all(app=app)
    app.run()
```

Jinja

http://jinja.pocoo.org/docs/2.10/templates/

- Jinja2 is a full featured template engine for Python
- Flask incorporates Jinja and helpers like `render_template`.
- It can format e-mails too.



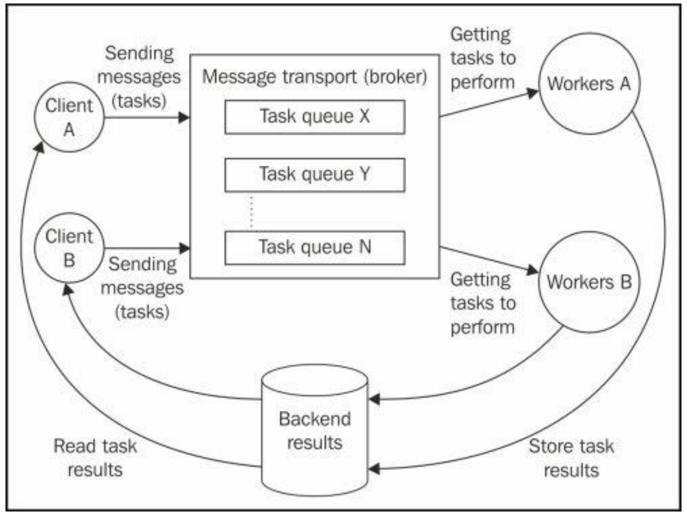
```
<html>
  <body>
    <h1>User List</h1>
    <ul>
      {% for user in users: %}
      <li>
      {{user.firstname}} {{user.lastname}}
      </li>
      {% endfor %}
    </ul>
  </body>
</html>
```

Background Tasks

Who is using celery

Instagram mozilla ADD-ONS AdRoll

- Celery is an **asynchronous task queue** based on distributed message passing.
- It is focused on real-time operations but supports scheduling as well.
- The execution units, called **tasks**, are executed concurrently on a single or more worker servers using multiprocessing.

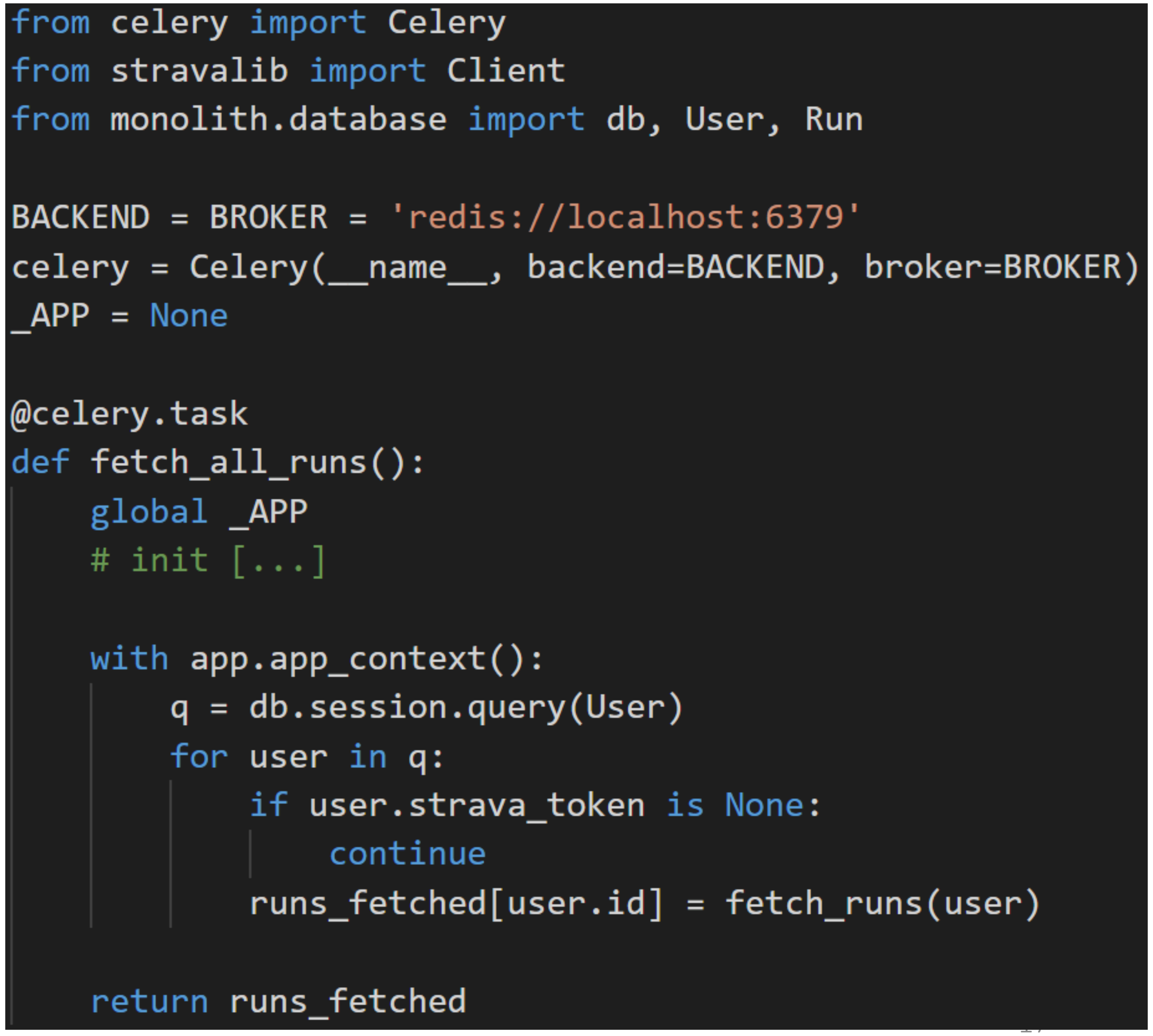


Celery

http://docs.celeryproject.org/en/latest/index.html

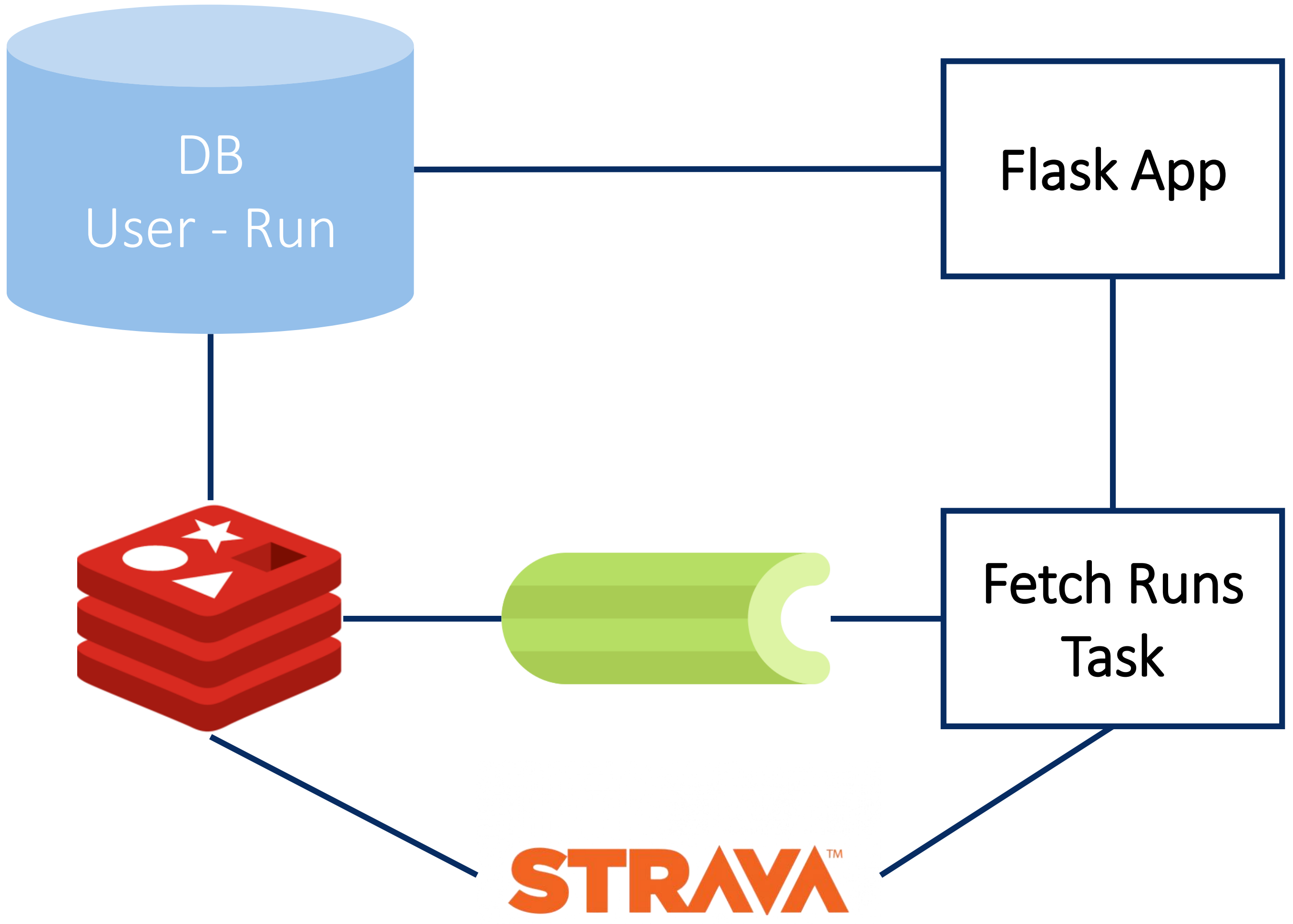
- The code that fetches runs from Strava can do this regularly, e.g. every hour.
- Background features run on their own **outside the request/response cycle** and use the SQLAlchemy models to do their job.
- An **intermediary message broker** oversees passing messages back and forth between the application and Celery. E.g.,

RabbitMQ redis amazon SQS



```python
from celery import Celery
from stravalib import Client
from monolith.database import db, User, Run

BACKEND = BROKER = 'redis://localhost:6379'
celery = Celery(__name__, backend=BACKEND, broker=BROKER)
_APP = None

@celery.task
def fetch_all_runs():
    global _APP
    # init [...]

    with app.app_context():
        q = db.session.query(User)
        for user in q:
            if user.strava_token is None:
                continue
            runs_fetched[user.id] = fetch_runs(user)

    return runs_fetched
```

The Monolith



- Currently, celery workers are triggered from the app via a request to `127.0.0.1/fetch`
- To make them periodic, have a look at Celery **Periodic Tasks** [http://docs.celeryproject.org/en/latest/userguide/periodic-tasks.html]