A small loan

whoami

Lorenzo Leonardini <@pianka>

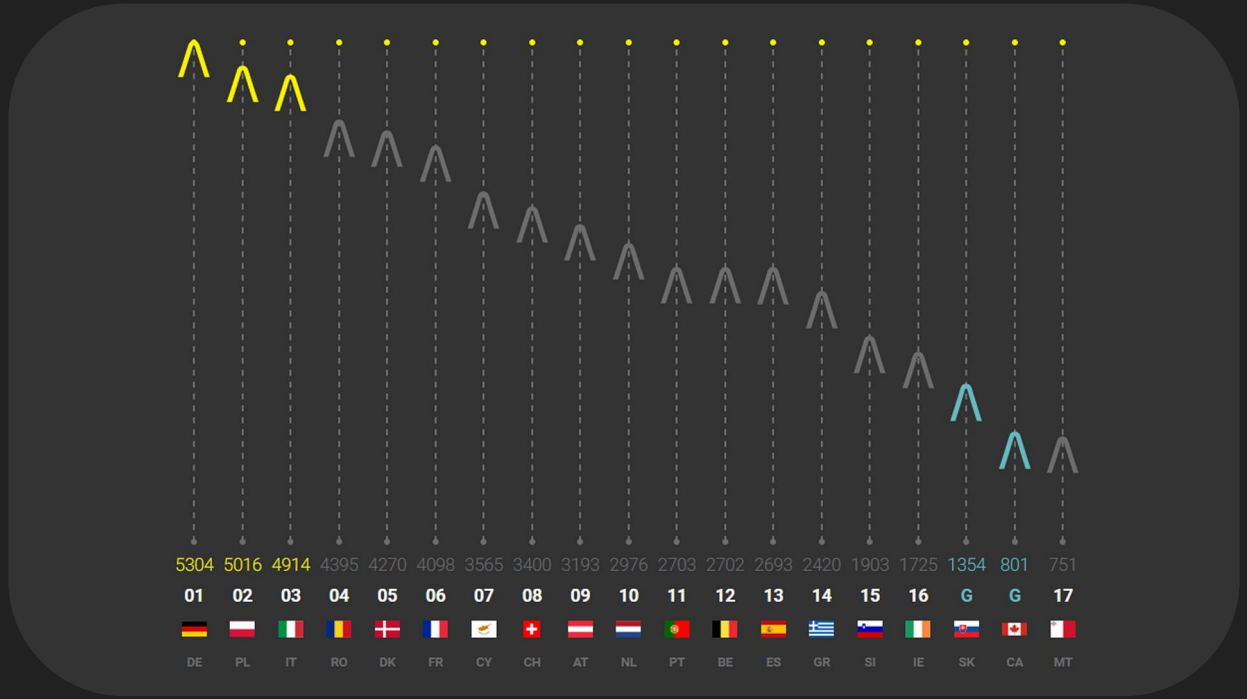
- web guy
- born in ZenHack
- about:blankets 
- TeamItaly 2018 & 2021
- boh che altro

ECSC 2021 - Praga

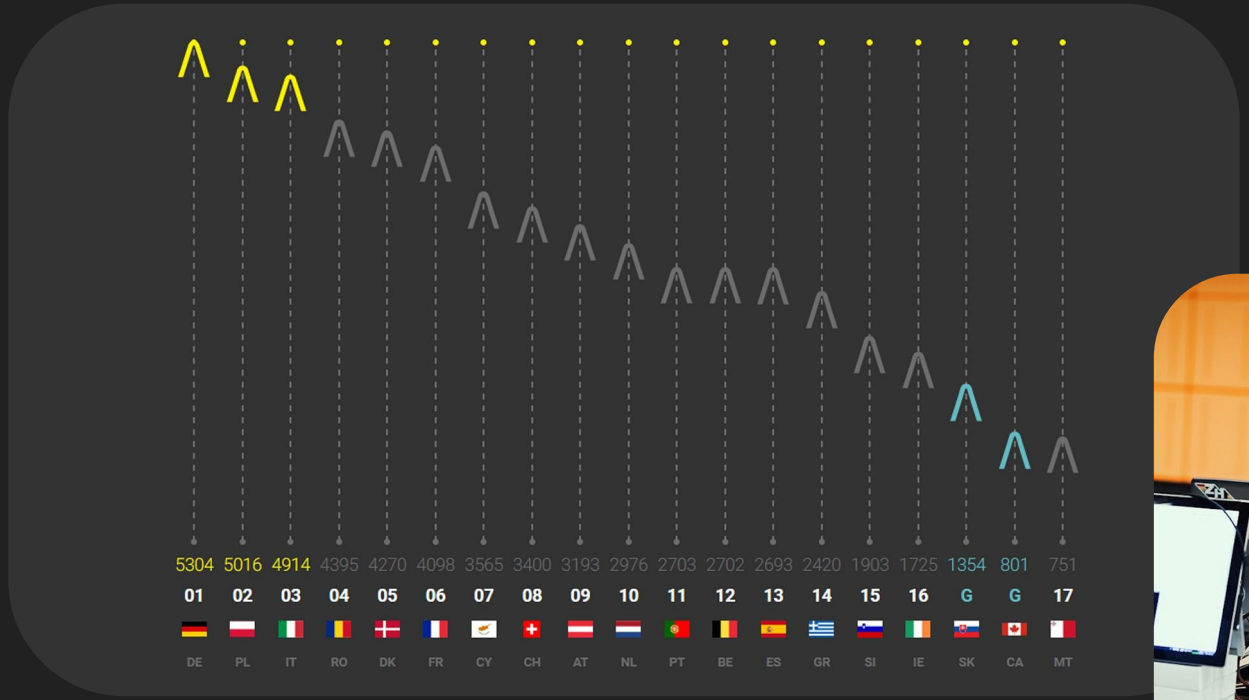
Two days

10:00 - 18:30

ECSC 2021 - Praga



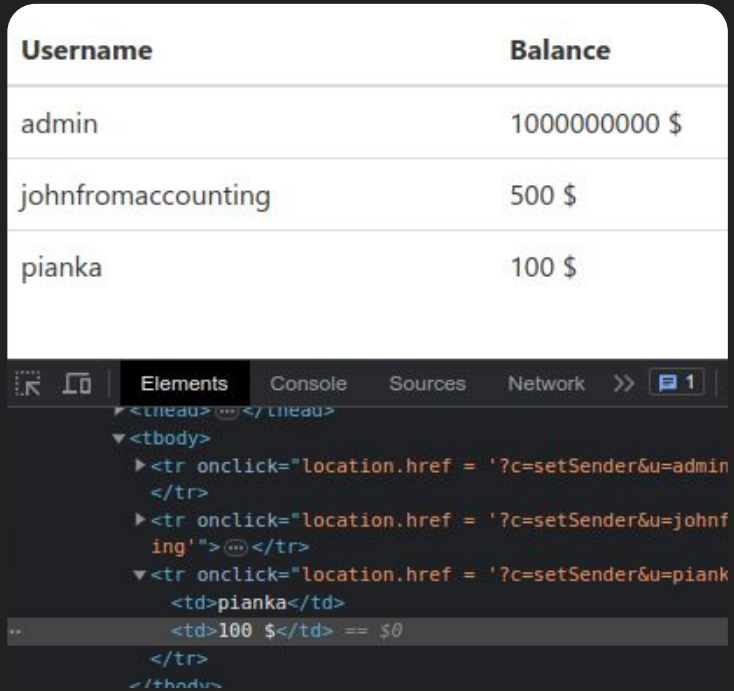
ECSC 2021 - Praga



ECSC 2021 - Praga

A small loan

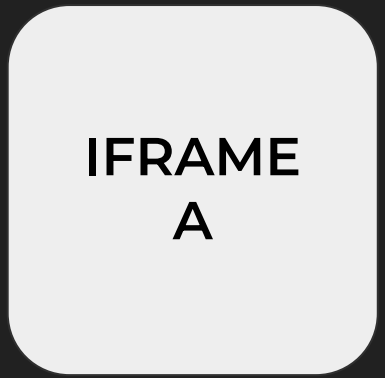
A small loan



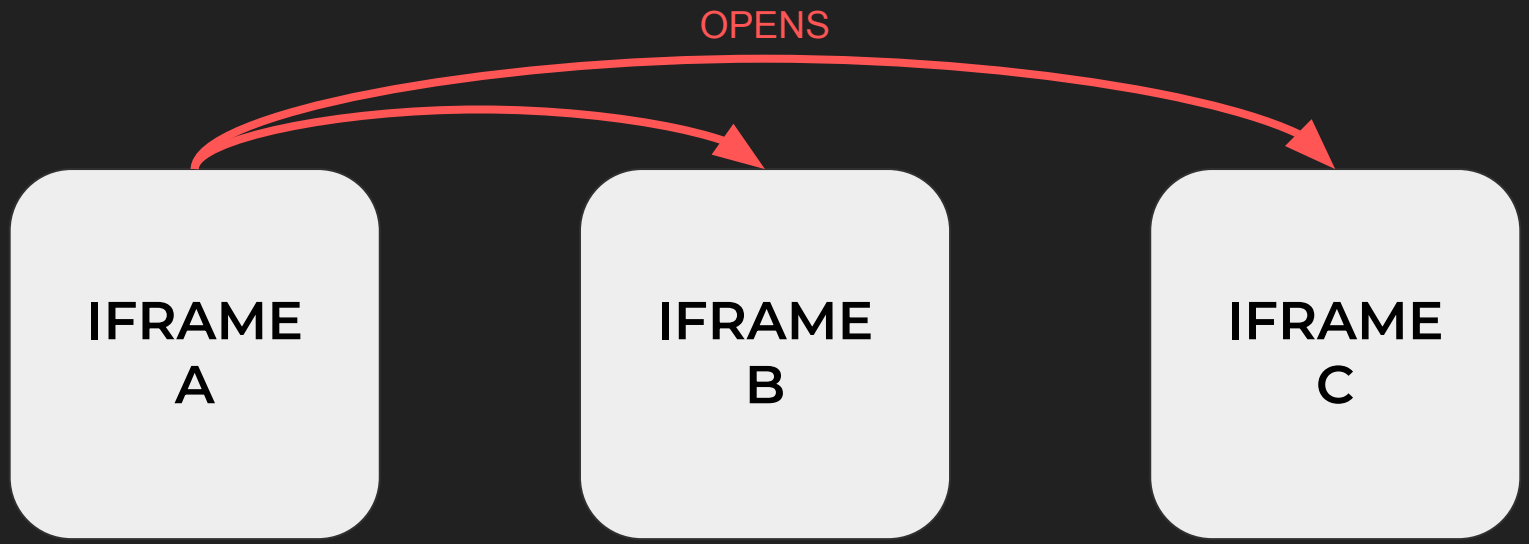

```
{% if callback %}
<script type="application/javascript">
    opener.{{ callback }}("{{ user }}")
</script>
{% endif %}
```

```
<script type="application/javascript">
    opener.setSender("pianka")
</script>
```

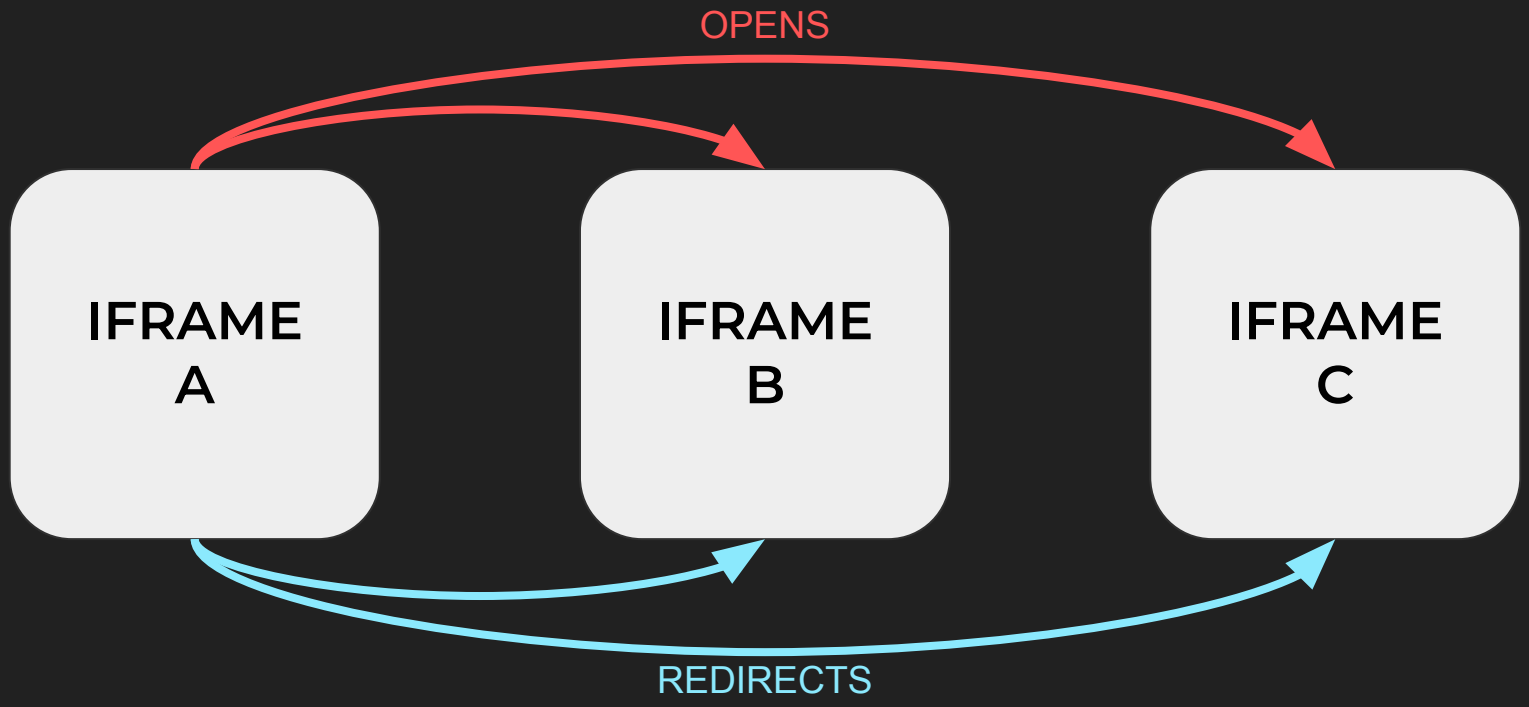
A small loan



A small loan



A small loan



A small loan

```
?c=document.body.firstElementChild.
nextElementSibling.nextElementSibling.
firstElementChild.nextElementSibling.
firstElementChild.nextElementSibling.
nextElementSibling.firstElementChild.
firstElementChild.nextElementSibling.
nextElementSibling.click
```

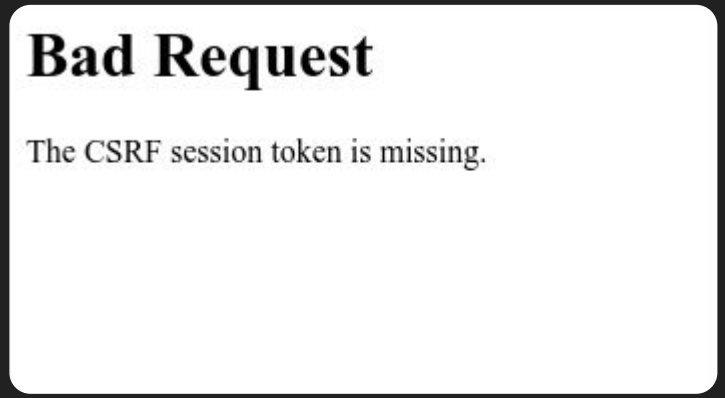
A small loan

```
recipientUser = User.query.get(requestFromDB.recipient)
senderUser = User.query.get(requestFromDB.sender)

if not senderUser.username == session.get('loggedInUser'):
    error = 'You cannot confirm this request'

if not (confirmed == "true" or confirmed == "false"):
    error="Request invalid"

if error:
    return redirect(url_for('viewMoneyRequest',id=id,e=error))

if confirmed == "true":
    if requestFromDB.amount > senderUser.balance:
        error = "Insufficient balance"

    if requestFromDB.amount > 1000000 and senderUser.username != "admin":
        error = "Only admin can make transfers larger then 1000000"

    senderUser.balance = (senderUser.balance-requestFromDB.amount)
    recipientUser.balance = (recipientUser.balance+requestFromDB.amount)

db.session.delete(requestFromDB)

if error:
    return redirect(url_for('viewMoneyRequest',id=id,e=error))

db.session.commit()
```

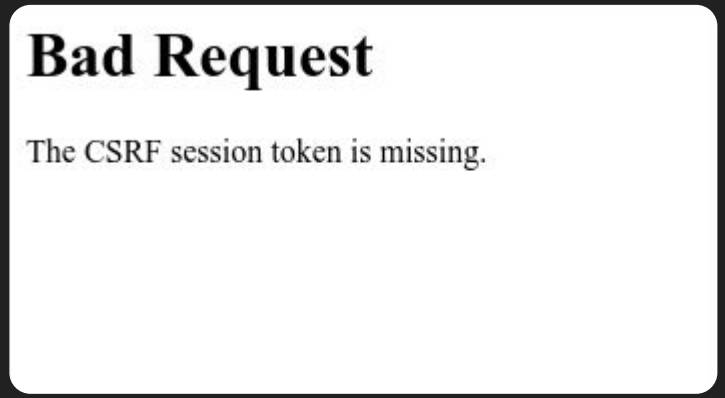
A small loan

```
if confirmed == "true":
    if requestFromDB.amount > senderUser.balance:
        error = "Insufficient balance"

    if requestFromDB.amount > 1000000 and senderUser.username != "admin":
        error = "Only admin can make transfers larger then 1000000"

    senderUser.balance = (senderUser.balance-requestFromDB.amount)
    recipientUser.balance = (recipientUser.balance+requestFromDB.amount)

db.session.delete(requestFromDB)

if error:
    return redirect(url_for('viewMoneyRequest',id=id,e=error))

db.session.commit()
```

A small loan



A small loan




```python
from flask_wtf.csrf import CSRFProtect

app.config['WTF_CSRF_CHECK_DEFAULT'] = False
csrf = CSRFProtect(app)

csrf.protect()
```

A small loan

Bad Request

The CSRF session token is missing.

```python
from flask_wtf.csrf import CSRFProtect

app.config['WTF_CSRF_CHECK_DEFAULT'] = False
csrf = CSRFProtect(app)

csrf.protect()
```

Flask WTF

Simple integration of Flask and WTForms, including CSRF, file upload, and reCAPTCHA.

A small loan

```
def generate_csrf(secret_key=None, token_key=None):
    """Generate a CSRF token. The token is cached for a request, so multiple
    calls to this function will generate the same token.

    During testing, it might be useful to access the signed token in
    ``g.csrf_token`` and the raw token in ``session['csrf_token']``.

    :param secret_key: Used to securely sign the token. Default is
        ``WTF_CSRF_SECRET_KEY`` or ``SECRET_KEY``.
    :param token_key: Key where token is stored in session for comparison.
        Default is ``WTF_CSRF_FIELD_NAME`` or ``'csrf_token'``.
    """
```

A small loan

```
if field_name not in g:
    s = URLSafeTimedSerializer(secret_key, salt='wtf-csrf-token')

    if field_name not in session:
        session[field_name] = hashlib.sha1(os.urandom(64)).hexdigest()

    try:
        token = s.dumps(session[field_name])
    except TypeError:
        session[field_name] = hashlib.sha1(os.urandom(64)).hexdigest()
        token = s.dumps(session[field_name])

    setattr(g, field_name, token)

return g.get(field_name)
```

A small loan

```
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: ebebf5bcfd34eb7f1de3925deacbabadfda16706
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: ebebf5bcfd34eb7f1de3925deacbabadfda16706
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
```

A small loan

```
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: ebebf5bcfd34eb7f1de3925deacbabadfda16706
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: ebebf5bcfd34eb7f1de3925deacbabadfda16706
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: None
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: None
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: None
g.csrf_token: ImViZWJmNWJjZmQzNGViN2YxZGUzOTI1ZGVhY2JhYmFkZmRhMTY3MDYi.ZJ7yNw.
session[csrf_token]: None
```

Flask contexts

Storing Data

The application context is a good place to store common data during a request or CLI command. Flask provides the `g object` for this purpose. It is a simple namespace object that has the same lifetime as an application context.

Note:

The `g` name stands for "global", but that is referring to the data being global *within a context*. The data on `g` is lost after the context ends, and it is not an appropriate place to store data between requests. Use the `session` or a database to store data across requests.

Flask contexts

```python
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

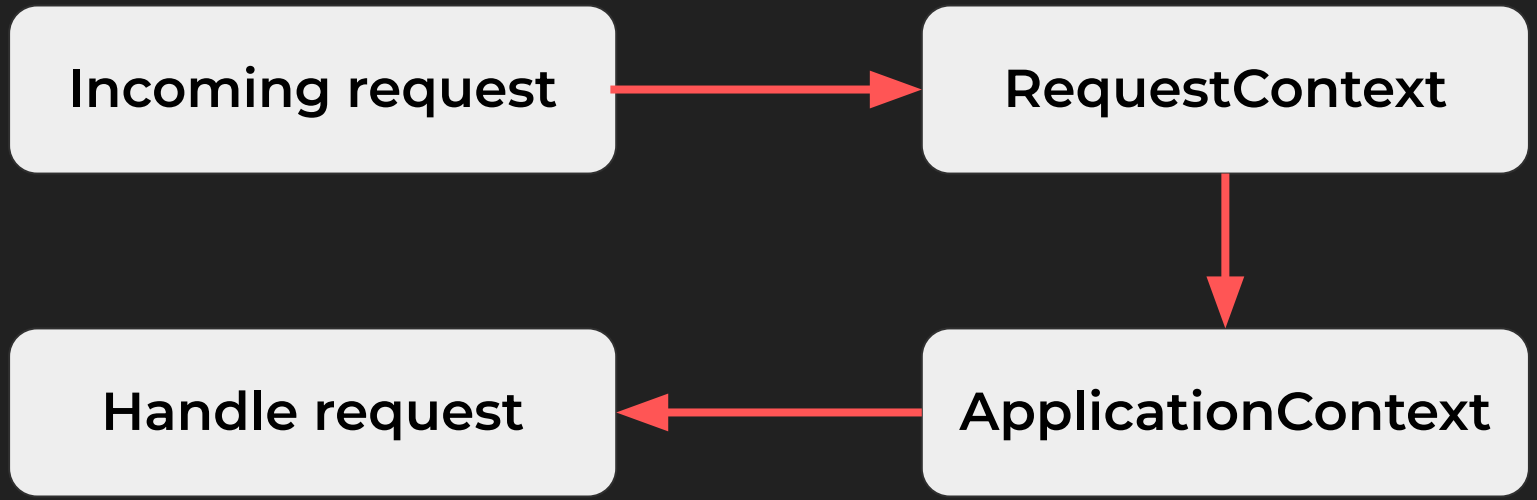
```javascript
server.route({
    method: 'GET',
    path: '/',
    handler: (request, h) => {

        return 'Hello World!';
    }
});
```

```javascript
app.get('/', (req, res) => {
    res.send('Hello World!')
})
```

Flask contexts (simplified)

```python
from flask import request, session, g

@app.get('/')
def home():
    user = request.args.get('user')
    return f'Hello {user}!'
```

Flask contexts (simplified)



A small loan

```
if field_name not in g:
    s = URLSafeTimedSerializer(secret_key, salt='wtf-csrf-token')

    if field_name not in session:
        session[field_name] = hashlib.sha1(os.urandom(64)).hexdigest()

    try:
        token = s.dumps(session[field_name])
    except TypeError:
        session[field_name] = hashlib.sha1(os.urandom(64)).hexdigest()
        token = s.dumps(session[field_name])

    setattr(g, field_name, token)

return g.get(field_name)
```

A small loan

```
if field_name not in g:
    s = URLSafeTimedSerializer(secret_key, salt='wtf-csrf-token')

    if field_name not in session:
        session[field_name] = hashlib.sha1(os.urandom(64)).hexdigest()

    try:
        token = s.dumps(session[field_name])
    except TypeError:
        session[field_name] = hashlib.sha1(os.urandom(64)).hexdigest()
        token = s.dumps(session[field_name])

    setattr(g, field_name, token)

return g.get(field_name)
```

A small loan

```
db.init_app(app)
app.app_context().push()

db.create_all()

for item in prepareInit():
    db.session.add(item)

db.session.commit()

app.config['WTF_CSRF_CHECK_DEFAULT'] = False
csrf = CSRFProtect(app)

redisConn = Redis(host='redis', port=6379, db=0)
q = Queue(connection=redisConn)

@app.route('/')
def main():
```

A small loan

```
db.init_app(app)
app.app_context().push()

db.create_all()

for item in prepareInit():
    db.session.add(item)

db.session.commit()

app.config['WTF_CSRF_CHECK_DEFAULT'] = False
csrf = CSRFProtect(app)

redisConn = Redis(host='redis', port=6379, db=0)
q = Queue(connection=redisConn)

@app.route('/')
def main():
```

A small loan

Create the Tables

After all models and tables are defined, call `SQLAlchemy.create_all()` to create the table schema in the database. This requires an application context. Since you're not in a request at this point, create one manually.

```
with app.app_context():
    db.create_all()
```

A small loan

Create the Tables

After all models and tables are defined, call `SQLAlchemy.create_all()` to create the table schema in the database. This requires an application context. Since you're not in a request at this point, create one manually.

```
with app.app_context():
    db.create_all()
```

3.x

In a nutshell, do something like this:

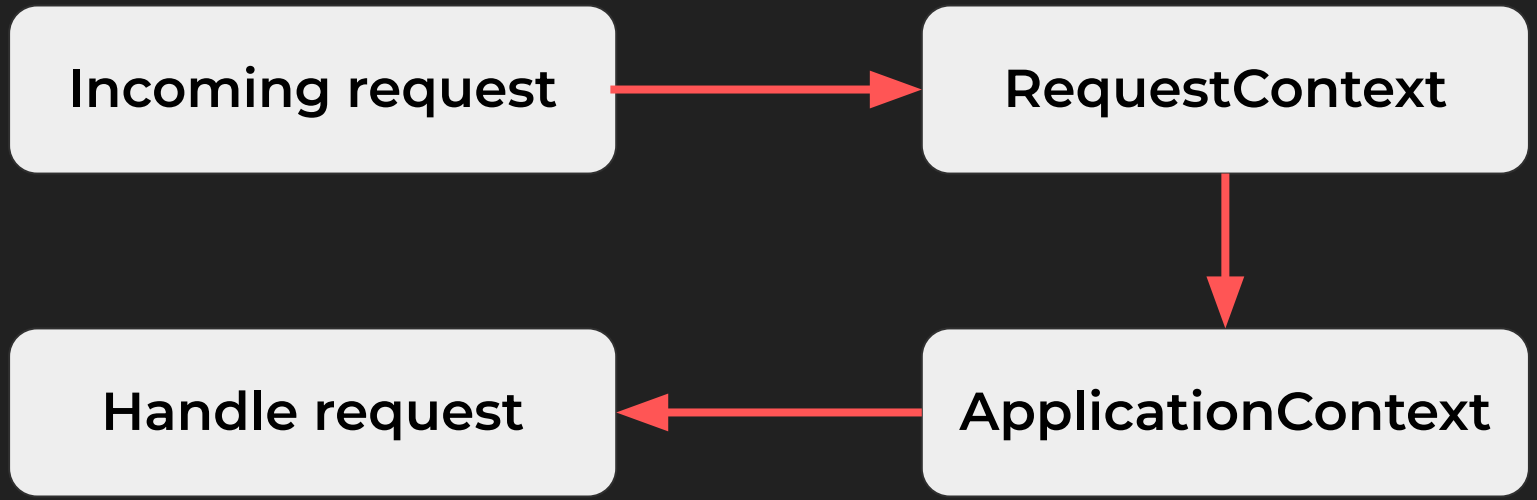
```
>>> from yourapp import create_app
>>> app = create_app()
>>> app.app_context().push()
```

Alternatively, use the with-statement to take care of setup and teardown:

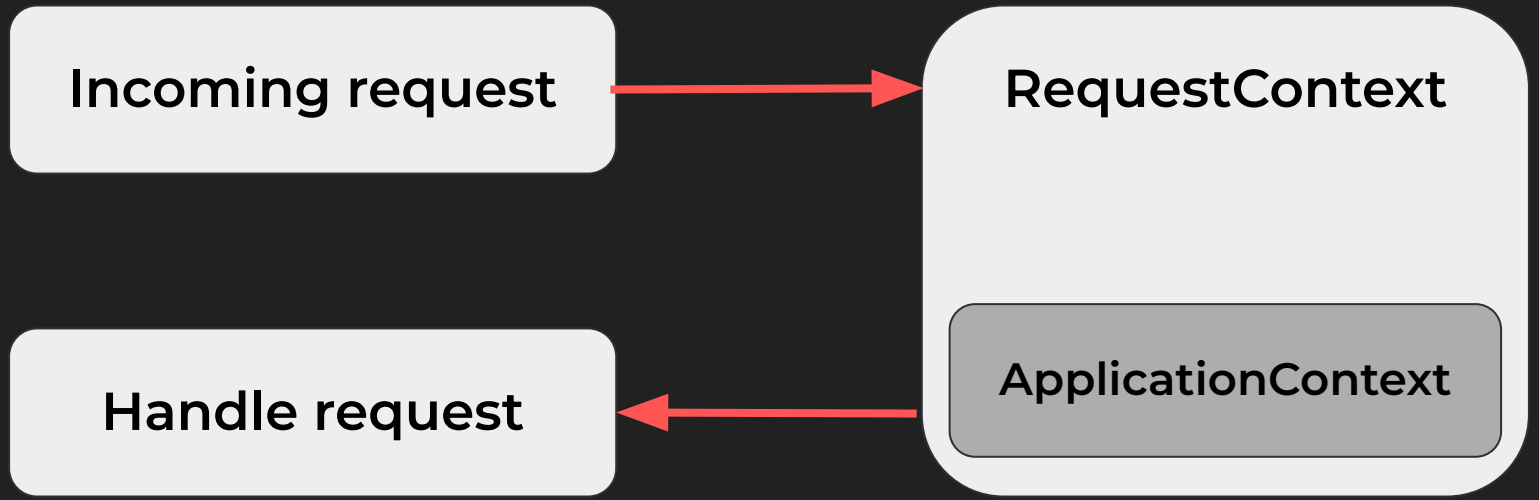
```
def my_function():
    with app.app_context():
        user = db.User(...)
        db.session.add(user)
        db.session.commit()
```

2.x

Flask contexts (simplified)

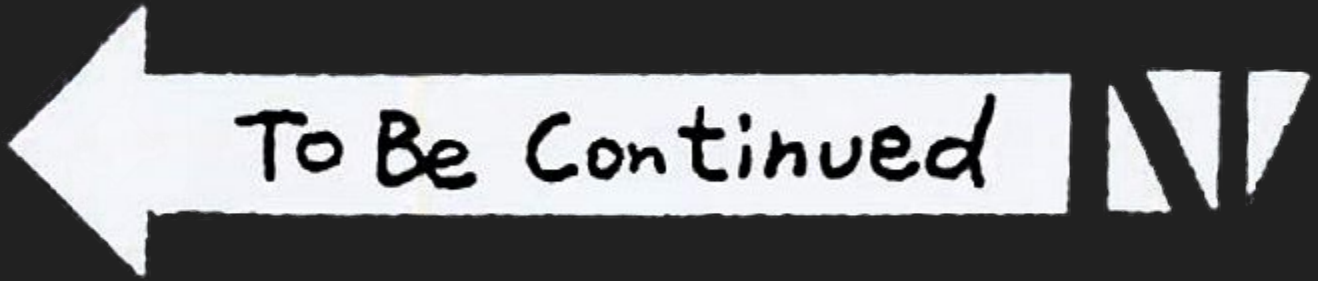


Flask contexts (simplified)



A small loan

```
FROM python:3.7-alpine
RUN apk add --no-cache gcc musl-dev build-base linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "--access-logfile", "-", "-w", "1", "-b", "0:5000", "app:app"]
#CMD ["python3", "app.py"]
```



https://sec.leonardini.dev

Lessons learnt


ABOUT:
BLANKETS