Flask Features
Flask Features
What We Covered
What We Didn't Cover
url_for
Blueprints
Signals
Lots of Jinja Stuff
Popular Add-Ons
WTForms & SQLA
WTForms-Alchemy
Flask-Login
Flask-Mail
Flask-Admin
Flask-Restless
Will You Use Flask?
Will You Use Flask?
Flask Versus ...
Flask v Node-Express
Flask v Django

Flask Wrap-Up

Springboard

Flask Features

A scalable, powerful, general-purpose web application framework.

What We Covered

- Routes
- Jinja templates
- Flask-SQLAlchemy
- Flask Testing
- Cookies & Sessions
- JSON and flask
- Flask-WTForms

We left a lot out — intentionally

What We Didn't Cover

url_for

you did this once...

```
@app.route("/users/<int:id>")
def user_profile(id): ...
```

you did this a dozen places...

```
<a href="/users/{{ user.id }}">See user</a>
```

and this in lots of places

```
def some_other_view():
    ...
    return redirect(f"/users/{user.id}")
```

What if you wanted to change that URL?

- Perhaps to ***/profiles/{user-id}***?
- Perhaps to ***/warbler/users/{user-ud}***?

you still do this once...

```
@app.route("/users/<int:id>")
def user_profile(id): ...
```

but now you don't need to hardcode URL

```
<a href="{{ url_for('user_profile', id=user.id) }}">go</a>
```

and this in lots of places

```
from flask import url_for

def some_other_view():
    ...
    redirect_url = url_for('user_profile', id=user.id)
    return redirect(redirect_url)
```

Blueprints

Build "applications" in Flask:

- Each app can have own models, forms, tests, views
- Can re-use an app in many sites
  - Many sites could use "blogly" app
- Useful for larger/more complex sites

Signals

"When [this thing] happens, do [this other] thing."

(eg send an email when a user registers, no matter how they register)

Lots of Jinja Stuff

Lots of additional features in Jinja:

- sharing parts of templates/repeated code
- formatting of numbers, dates, lists in the template
- caching parts of templates ("this part only changes every 5 mins")
- and more!

Popular Add-Ons

WTForms & SQLA

you did this a lot

```
def edit_pet(pet_id):
    pet = Pet.query.get(pet_id)
    form = EditPetForm(obj=pet)

    if form.validate_on_submit():
        pet.name = form.name.data
        pet.species = form.species.data
        pet.color = form.color.data
        pet.age = form.age.data
        pet.weight = form.weight.data
        pet.num_legs = form.num_legs.data
        ...
```

you can do this

```
def edit_pet(pet_id):
    pet = Pet.query.get(pet_id)
    form = EditPetForm(obj=pet)

    if form.validate_on_submit():
        form.populate_obj(pet)
```

WTForms-Alchemy

Can generate WTForms from SQLAlchemy model:

forms.py

```
from flask_wtf import FlaskForm
from wtforms_alchemy import model_form_factory
from models import db, Pet, Owner

BaseModelForm = model_form_factory(FlaskForm)

class ModelForm(BaseModelForm):
    @classmethod
    def get_session(self):
        return db.session

class PetForm(ModelForm):
    class Meta:
        model = Pet

class OwnerForm(ModelForm):
    class Meta:
        model = Owner
```

Flask-Login

Product that provides common parts of user/passwords/login/logout

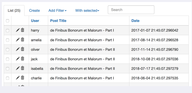
Similar to what you built, but out-of-box.

Flask-Mail

Can send email from Flask!

Flask-Admin

Can get decent CRUD admin views from SQLAlchemy models:



Flask-Restless

Get CRUD API endpoints from SQLAlchemy models:

```
from flask.restless import APIManager

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.Unicode)
    birth_date = db.Column(db.Date)

class Computer(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.Unicode)
    vendor = db.Column(db.Unicode)
    purchase_time = db.Column(db.DateTime)
    owner_id  db.Column(db.Integer, db.ForeignKey('user.id'))
    owner = db.relationship('Person')

# Create the Flask-Restless API manager.
manager = APIManager(app, flask_sqlalchemy_db=db)

# API endpoints, available at /api/<tablename>
manager.create_api(User, methods=['GET', 'POST', 'DELETE'])
manager.create_api(Computer, methods=['GET'])
```

Will You Use Flask?

Maybe.

It's popular and used by real companies, large and small.

It's also a great choice for personal projects, code challenges, etc.

Flask Versus ...

Flask v Node-Express

Pretty similar, actually.

Both work at same "level of concepts", and share lots of ideas.

You can even use Jinja to make templates with Express!

Flask v Django

Django is a popular, larger, more featureful Python Framework.

It's a *higher level* and *more opinionated*

Flask ***model.py***

```
class Pet(db.Model):
    id = ...
    name = ...
    color = ...
    owner_id = ...

    owner = db.relationship("Owner", backref="pets")
```

Django ***model.py***

```
class Pet(models.Model):
    name = ...
    color = ...
    owner = models.ForeignKey("Owner")

    # assumes "id" of auto-incrementing int
    # defines relationship & make "owner_id" column
```

Flask ***app.py***

```
@app.route("/pets/<int:id>/edit", methods=["GET", "POST"])
def edit_pet(id):
    """Show pet edit form / handle edit."""

    pet = Pet.query.get(id)
    form = PetEditForm(obj)    # need to make form!

    if form.validation_on_submit():
        pet.name = form.name.data
        pet.color = form.color.data
        db.session.commit()
        redirect(f"/pets/{id}")

    return render_template("pet_edit.html", form=form)
```

Django ***views.py***

```
class PetEditView(generic.UpdateView):
    """Show pet edit form / handle edit."""

    model = Pet
```

So, is Django "better"?

Nope.

If you like Django's patterns & they fit your use cases, you can build an app faster by following those patterns.

But:

- they take a longer amount of time to learn
- if things break, it can be harder to understand
- if you want to change how things work, it can be harder

Flask is like a really nice bicycle:

It's great for easy trips, can scale up to long journeys, isn't too opinionated about where you use it, and it's relatively easy to understand and fix.