

# Blogly

[Download exercise <../flask-blogly.zip>](#)

This is a two-day exercise to practice SQLAlchemy with relationships.

In it, you'll build “Blogly”, a blogging application.

## Part One

### Installing Tools

```
(env) $ pip install psycopg2-binary
(env) $ pip install flask-sqlalchemy
```

### Create User Model

User
<b>id [PK]</b> first_name last_name image_url

First, create a **User** model for SQLAlchemy. Put this in a **models.py** file.

It should have the following columns:

- **id**, an autoincrementing integer number that is the primary key
- **first\_name** and **last\_name**
- **image\_url** for profile images

Make good choices about whether things should be required, have defaults, and so on.

### Create Flask App

Next, create a skeleton Flask app. You can pattern match from the lecture demo.

It should be able to import the **User** model, and create the tables using SQLAlchemy. Make sure you have the FlaskDebugToolbar installed — it's especially helpful when using SQLAlchemy.

### Make a Base Template

Add a base template with slots for the page title and content. Your other templates should use this.

You can use Bootstrap for this project, but don't spend a lot of time worrying about styling — this is **not** a goal of this exercise.

## User Interface

Here is what you should build:

### User Listing

# Users

- [Alan Alda](#)
- [Joel Burton](#)
- [Jane Smith](#)

Add user

<\_images/list.png>

### New User Form

# Create a user

First Name

Enter a first name

Last Name

Enter a last name

Image URL

Provide an image of this user

Add

<\_images/new.png>

### User Detail Page



# Joel Burton

Edit

Delete

<\_images/detail.png>

### User Edit Page

# Edit a user

First Name

Last Name

Image URL

[<\\_images/edit.png>](#)

## Make Routes For Users

### Note: We Won't Be Adding Authentication

While this application will have “users”, we’re not going to be building login/logout, passwords, or other such thing in this application. Any visitor to the site should be able to see all users, add a user, or edit any user.

Make routes for the following:

#### GET /

Redirect to list of users. (We’ll fix this in a later step).

#### GET /users

Show all users.

Make these links to view the detail page for the user.

Have a link here to the add-user form.

#### GET /users/new

Show an add form for users

#### POST /users

Process the add form, adding a new user and going back to **/users**

#### GET /users/[user-id]

Show information about the given user.

Have a button to get to their edit page, and to delete the user.

#### GET /users/[user-id]/edit

Show the edit page for a user.

Have a cancel button that returns to the detail page for a user, and a save button that updates the user.

### POST `/users/[user-id]/edit`

Process the edit form, returning the user to the `/users` page.

### POST `/users/[user-id]/delete`

Delete the user.

## Get a Code Review

**Warning: STOP AND GET A CODE REVIEW.**

We really want to see your code at this point, before you continue onto the other steps.

## Part One: Further Study

There are two more big parts to this exercise—but if you feel like you’re ahead of the group, here is some further study for this part you can work on.

### Add Full Name Method

It’s likely that you refer to users by `{{ user.first_name }}` `{{ user.last_name }}` in several of your templates. This is mildly annoying to have to keep writing out, but a big annoyance awaits: what would happen if you added, say, a ***middle\_name*** field? You’d have to find & fix this in every template.

Better would be to create a convenience method, ***get\_full\_name()***, which you could use anywhere you wanted the users’ full name:

```
>>> u = User.query.first()

>>> u.first_name      # SQLAlchemy attribute
'Jane'

>>> u.last_name       # SQLAlchemy attribute
'Smith'

>>> u.get_full_name()
'Jane Smith'
```

Write this.

Change your templates and routes to use this.

### List Users In Order

Make your listing of users order them by ***last\_name***, ***first\_name***.

You can have SQLAlchemy do this—you don’t need to do it yourself in your route.

## Turn Full Name Into a “Property”

Research how to make a Python “property” on a class — this is something that is *used like* an attribute, but actually is a method. This will let you do things like:

```
>>> u = User.query.first()

>>> u.first_name      # SQLAlchemy attribute
'Jane'

>>> u.last_name       # SQLAlchemy attribute
'Smith'

>>> u.full_name       # "property"
'Jane Smith'
```

## Part Two

[Continue to Part Two <two.html>](#)

## Solution

[Our Solution for Part One <solution/index.html>](#)