Lecture 14: Intro to Django, Model, Admin

The Big Picture

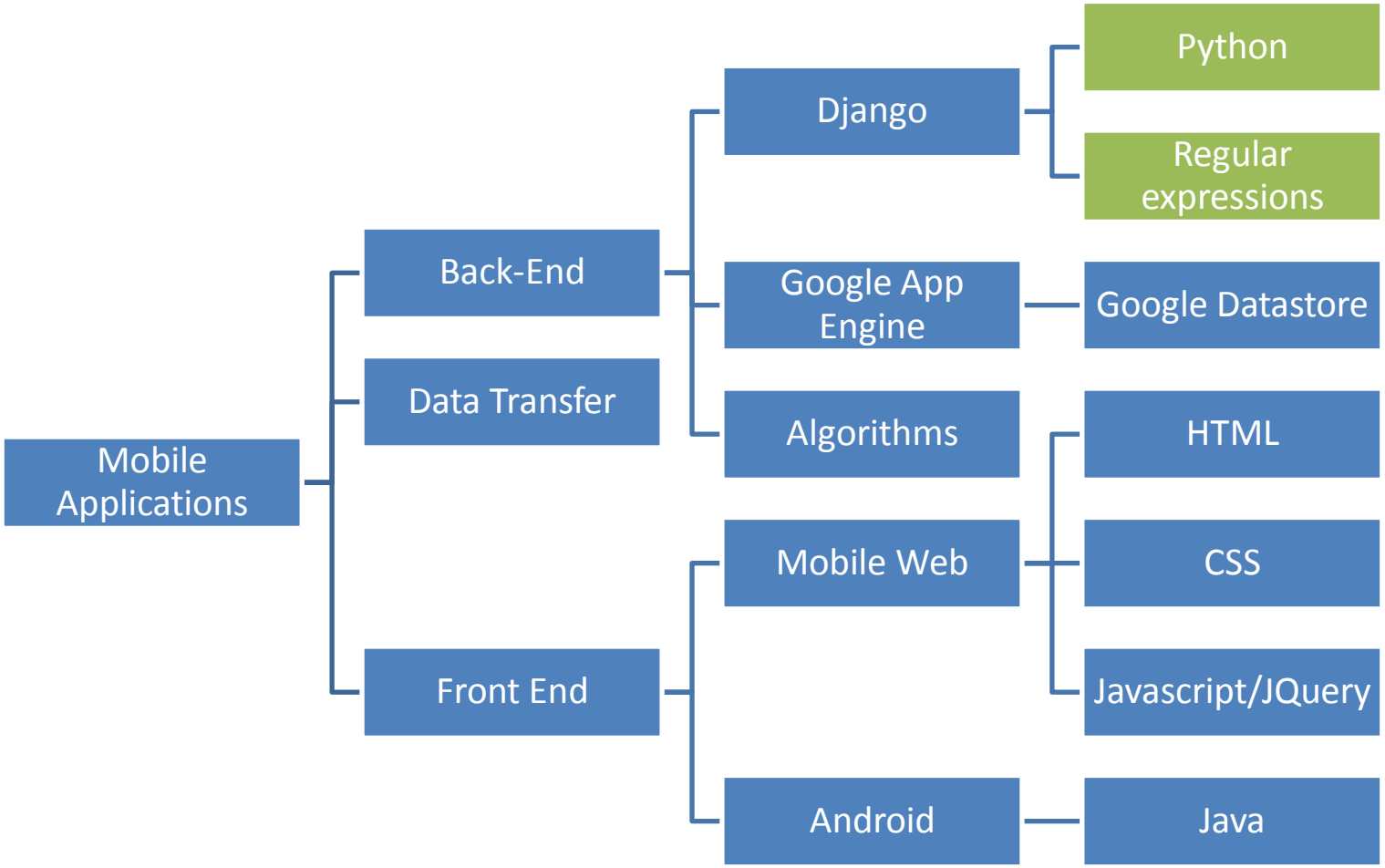
Google App Engine

Your Django app

Android OS

Your Android app

Course Roadmap



Development Tools

- Operating system
  - Windows (Optimally Linux)
- Integrated Development Environment
  - Eclipse (Pydev)
- Version Control
  - git, GitHub

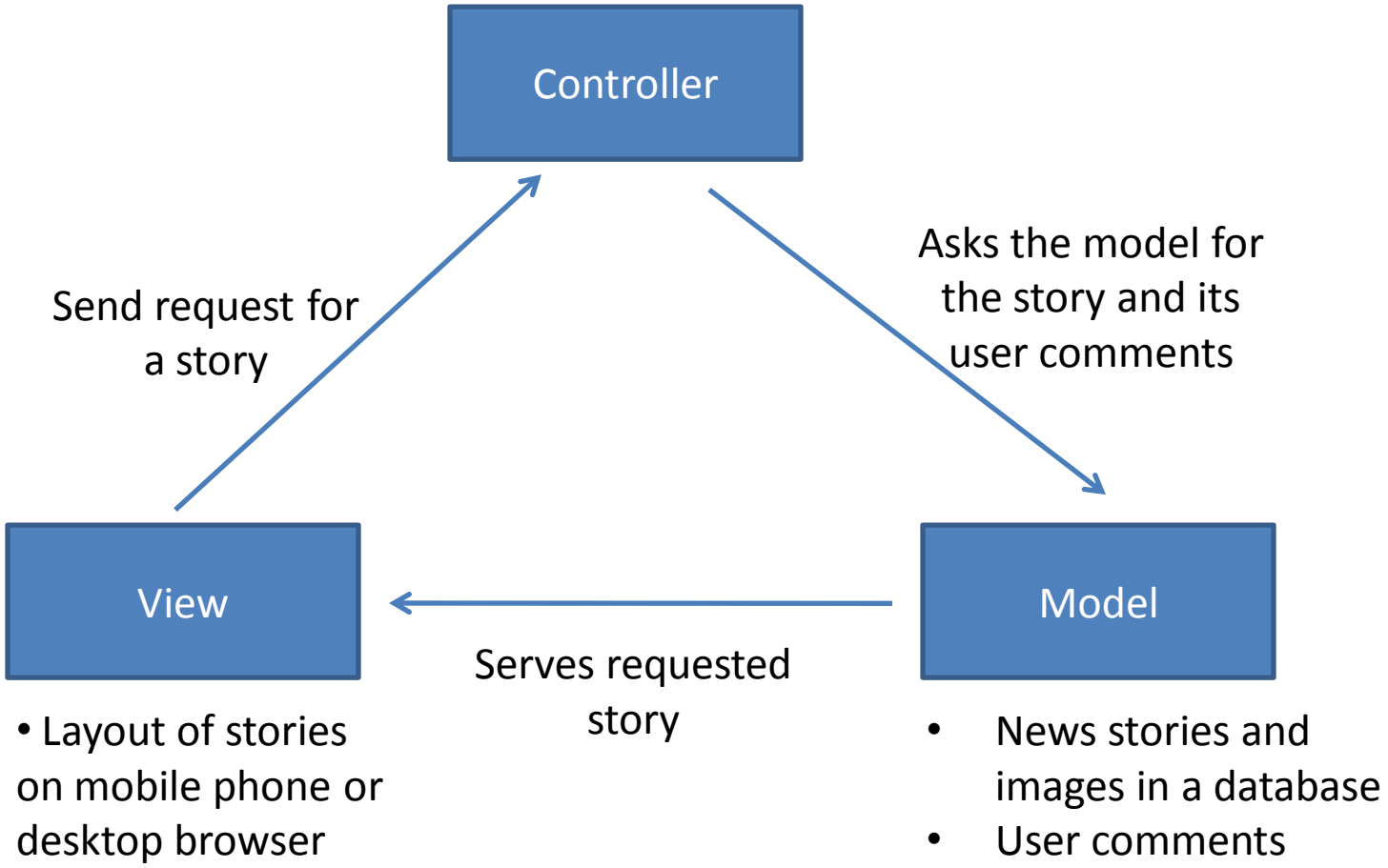
Web Application Framework

- A framework (a.k.a. code libraries) that provides functionality for common components in a website, web app, or web service.
- Eases coding for
  - Working with forms
  - Handling HTTP requests
  - Templates for common HTML layouts
  - URL mapping
  - Database communication
  - Session management
  - Site security
- Allows you to focus on design and functionality rather than small details.

Model-View-Controller (MVC)

- A paradigm for organizing code often seen in web app frameworks
- Main idea is
  1. Separate the storage and manipulation of data (the model) and the presentation of data (view)
  2. Use the Controller to communicate between the model and view
- Advantages
  - Easier to develop and test model and view independently
  - Easier for others to understand
- Exact roles of model, view, and controller depend on who you ask!

Model-View-Controller (MVC) (news site example)



Google App Engine

- Google's cloud computing platform to develop and host web applications
- Distributed data storage service (The Datastore)
- Free up to 500 MB of storage and 5 million page views
- Saves the hassle and initial costs of setting up your own server equipment and software
- Supports Java and Python

What is Django?

- Web application framework, written in Python
- Released 2005
- Began with World Online, that needed to rapidly develop applications for news sites.
- Named after gypsie jazz guitarist Django Reinhardt (1910-53)
- Follows the Model-View-Controller paradigm

Why Django?

- Fast and easy development of web applications
  - Modular and re-useable. Don't Repeat Yourself (DRY) principle
  - Hides database details
- Active development and wide community support
- Successful Django sites http://djangosites.org/
- Supported by Google App Engine

Setting up your Django DB

- We will be using sqllite 3 … it is bundled with Django, no installations required
- In your settings.py , modify
  Engine : django.db.backends.sqlite3
  Name : C:\pyprojects\mysite\db
- Run python manage.py syncdb to create db required by your imported libraries.
- More on

  https://docs.djangoproject.com/en/1.4/intro/tutorial01/

Creating a Django App within a Project

- An app is a Web application that does something -- e.g., a Weblog system, a database of public records or a simple poll app. A project is a collection of configuration and apps for a particular Web site. A project can contain multiple apps. An app can be in multiple projects.
- Always add the app name to settings.py to inform it about the apps existence

What is a model?

- A class describing data in your application
- Basically, a class with attributes for each data field that you care about
- The schema for your data

Django models

- Avoid direct work with the database
- No need to handle database connections, timeouts, etc. Let Django do it for you.
- Class that extends models.Model

Django fields

- All you do is define a field type
  - Ex: active = models.BooleanField()
- Django handles the rest:
  - Bit value in sql database
  - Represented as a checkbox on a webpage
  - Validation of values

Django Model Syntax

```
class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)
    def __unicode__():
        return last_name+” , “+first_name

class Album(models.Model):
    artist = models.ForeignKey(Musician)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
    def __unicode__():
        return name
```

Django Model Syntax

- class Album(models.Model): artist = models.ForeignKey(Musician) name = models.CharField(max_length=100) release_date = models.DateField() num_stars = models.IntegerField()

Important Django field types

- BooleanField
  - Checkbox
- CharField(max_length)
  - Single-line textbox
- DateField
  - Javascript calendar
- DateTimeField
  - Javascript calendar, time picker

Important Django field types

- DecimalField(max_digits, decimal_places)
  - Decimal numbers
- EmailField
  - Charfield that validates email address
- FileField
  - File upload, stores path in database
- FloatField
  - Floating point numbers

Important Django field types

- ImageField ***Don’t use
  - Stores images
- IntegerField
  - Integer textbox
- PositiveIntegerField
  - Integer textbos for positive integers
- TextField
  - Multi-line textbox

Important Django Field types

- TimeField
  - Time picker
- URLField
  - Textbox for URLs
- Anything you create

Field options

- Null
- Blank
- Choices:
  - List or tuple of 2-tuples to use as field choices
  - Django will represent it with a drop-down instead of a textbox
- Default
- Help text

More field options

- Primary key
- unique
- Verbose field name

DateField and DateTimeField options

- Auto_now
  - Any time the object is saved, the field will be updated with the current time.
- Auto_now_add
  - The time will always be equal to the creation date of the object.

Model Methods

- __unicode__():
  - Equivilant of toString – used for auto-generated admin pages
- Get_absolute_url()
  - Used for deciding URLs that reference a specific object

Django Relationship Fields

- ForeignKey(foreign class)
  - Many-to-one
- ManyToManyField(foreign class)
  - Uses a temporary table to join tables together
- OneToOneField(foreign class)
  - Enforces uniqueness

Rules of Django Models

1. When you update a model, ALWAYS RUN

```
python manage.py syncdb
```

2. Keep code clean
3. Always create a __unicode__() method
4. Name your variables well
5. Don’t think too much about the database

Commands

- View sql for models in a webapp

python manage.py sql appname

- Create the tables in database

python manage.py syncdb

Add an App to Admin Interface

- Create admin.py in your appname directory

```
from appname.models import Tablename
from django.contrib import admin
admin.site.register(Tablenames)
```

Django Admin

- Used for inputting, editing, and deleting data from your application
- Saves you from manually creating admin forms
- Automatically generated based on your models
- Customizable through admin.py

Most important rule

- Django **admin**
- Not Django user

Django Admin Nevers

- Never:
  - Give normal users access to django admin
  - Give anybody access that you don’t 100% trust

Blog example

- If you are the only blogger, you can use the admin interface
- If you provide a blogging service, you need to make a user interface

- You MUST create a separate interface for users to add comments
- You can use Django Admin to clean up comments

Admin pages

- Home (All Models that are registered)
  - List (all objects of that Model)
    - Details (all attributes of that object)

Ex:

- Home
  - Blogs
    - Blog post

Default admin

```
class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()
    def __unicode__(self):
        return self.title

admin.site.register(Book)
```

Extended Admin

```
class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()
    def __unicode__(self):
        return self.title

class BookAdmin(admin.ModelAdmin):
    pass

Admin.site.register(Book, BookAdmin)
```

Extended Admin Example

```
Class BookAdmin(admin.ModelAdmin):
    list_display = ( ‘title’ ,’ publisher’ ,
        ’ publication_date’ )
    list_filter = ( ‘publisher’ ,
        ’ publication_date’ )
    search_fields = ( ‘title’ , ‘publisher’ )
    ordering = ( ‘title’ ,’ -publication_date’ )
```

Extending your model

- Goal: display the first 10 letters of a book title
- Solution:

In your model, create a method:

```
def title_first_10(self):
    return self.title[:10]
```

In the admin class, add:

```
list_display = ( 'title_first_10' )
```

Inlines

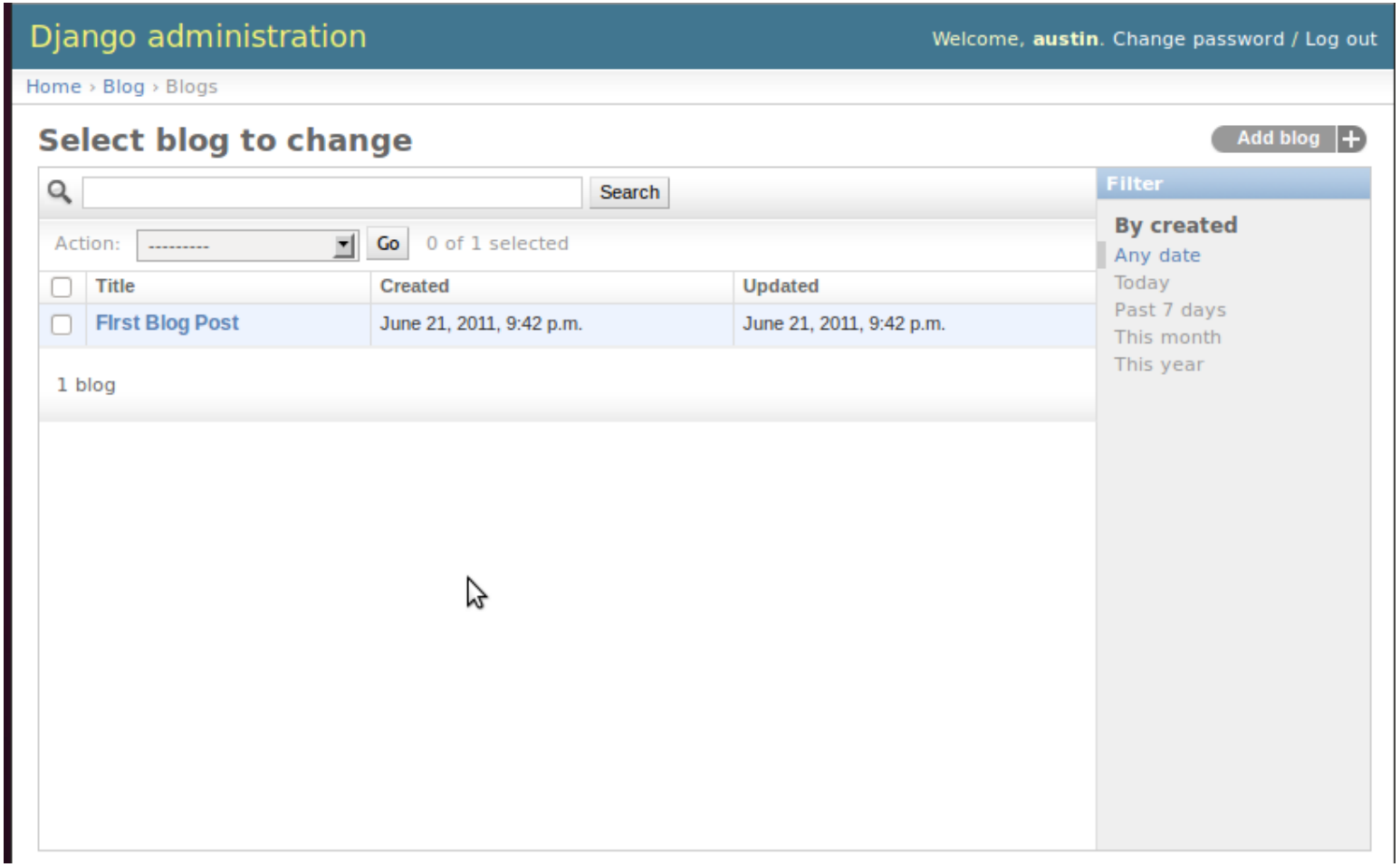
- On the admin pages, you may want to see all the Book objects that relate to one Author.
- Django Admin lets you put this all on one page with minimal effort

Inline Syntax

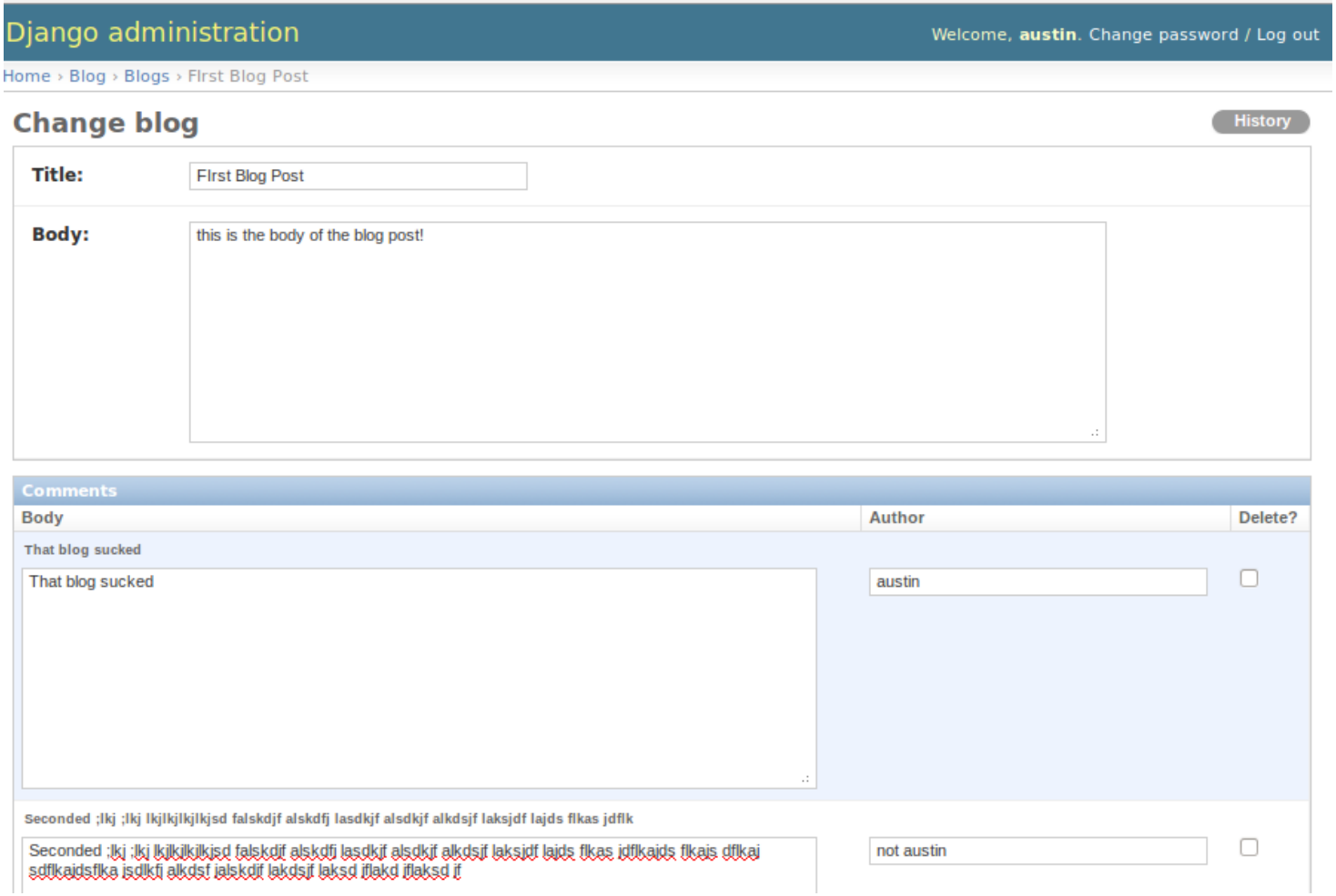
```
class BookInline(admin.TabularInline):
    model = Book
class AuthorAdmin(admin.ModelAdmin):
    inlines = [BookInline]
```

(You can use either TabularInline or StackedInline)

We want this:


Django administration
Welcome, austin. Change password / Log out
Home › Blog › Blogs
Select blog to change
Add blog
Search
Action: ---------
Go
0 of 1 selected
Title
Created
Updated
First Blog Post
June 21, 2011, 9:42 p.m.
June 21, 2011, 9:42 p.m.
1 blog
Filter
By created
Any date
Today
Past 7 days
This month
This year

And this:


Django administration
Welcome, austin. Change password / Log out
Home › Blog › Blogs › FIrst Blog Post
Change blog
History
Title:
FIrst Blog Post
Body:
this is the body of the blog post!
Comments
Body
Author
Delete?
That blog sucked
That blog sucked
austin
Seconded ;lkj ;lkj lkjlkjlkjlkjsd falskdjf alskdfj lasdkjf alsdkjf alkdsjf laksjdf lajds flkas jdflk
Seconded ;lkj ;lkj lkjlkjlkjlkjsd falskdjf alskdfj lasdkjf alsdkjf alkdsjf laksjdf lajds flkas jdflkajds flkajs dflkaj sdflkajdsflka jsdlkfj alkdsf jalskdjf lakdsjf laksd jflakd jflaksd jf
not austin