INFS3202/7202 – Web Information Systems

Lecture Week 12: Transitioning to Other Web Frameworks

Dr Aneesha Bakharia (Senior Lecturer, EECS)
Email: a.bakharia1@uq.edu.au

Contents

Course Updates

Issue/Feedback	Change
SECaTS Open Today	• Please provide feedback on INFS3202/7202
Project Assessment Item	• Due Friday Week 12 at 3pm • 1 week remaining

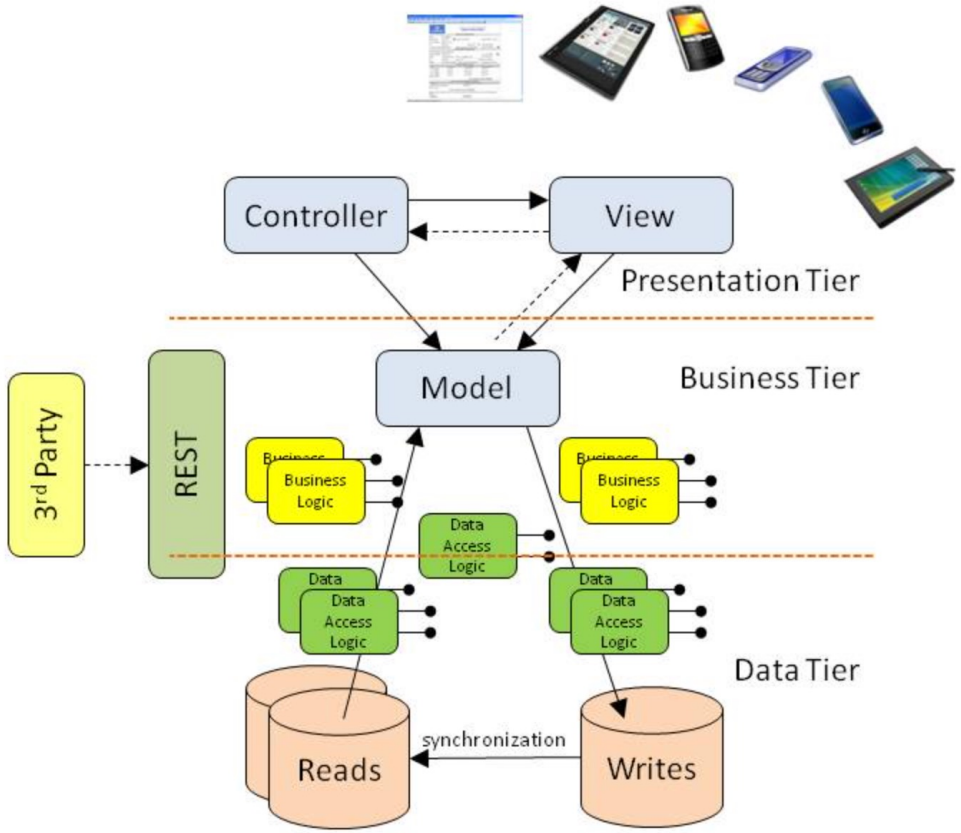
Learning Objectives

From Course Profile

1. Apply system architecture principles to design and deploy Web Information Systems (WIS) solutions.
2. Evaluate and articulate the scope, complexity, and key considerations in the design and implementation of Web Information Systems.
3. Design and program Web Information Systems (WIS) with server-side functionalities.
4. Develop responsive Web-based, database-driven applications using efficient and effective technologies.
5. Evaluate and justify the suitability of Web Information Systems solutions in various contexts, considering factors such as user needs and technical constraints.
6. Judge in which situations WIS solutions are more or less appropriate.
7. Critically analyze current issues and emerging trends in Web Information Systems development, and predict potential impacts on future practices and technologies.

Recap of CodeIgniter MVC Concepts

Model View Controller (MVC)



The MVC pattern is a specialized form of 3-layer architecture, making it highly relevant to modern web development practices

Image from https://en.wikiversity.org/wiki/Three-Tier_Architecture#/media/File:MVC3tierArchitecture.jpg

Convention vs Configuration

Convention vs. Configuration is a design philosophy that:

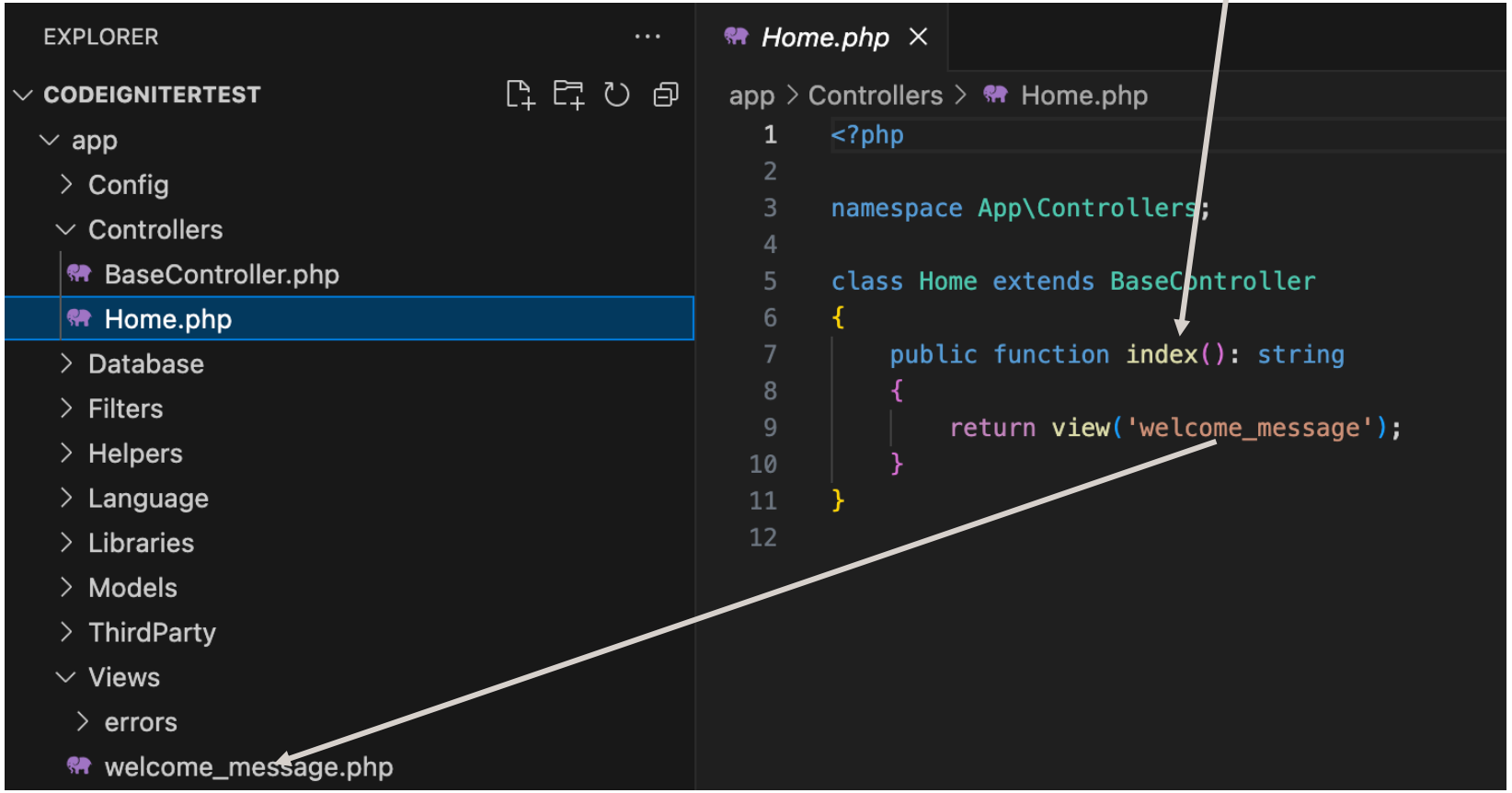
- impacts how software frameworks are structured (i.e. where files are located)
- how they expect developers to interact with them

- It's a trade-off between the ease of following predefined paths and files (convention) or the flexibility to define everything explicitly (configuration).

- Important advice:
  - Web frameworks (CodeIgniter, Django, Ruby on Rails, Flask, Next,JS) come with a predefined folder structure
  - Learn what the structure is and where files go
  - This will improve your productivity!

How is the CI 4.4 Welcome Message displayed?

http://localhost:8080



- / path runs the index() method
- the view function specifies the .PHP file to run
- The .PHP file can contain HTML and embedded PHP
- Welcome_message.php contains the “Welcome to CI 4.4 message”

- Flow of connections between files is very important! Understanding this is very important.

CodeIgniter – Mapping a Model to a DB Table

- **Model Configuration: The model class provides configuration options to facilitate the smooth functioning of its methods.**
  - **$table**: Specifies the database table that this model primarily works with.
  - **$primaryKey**: Specifies the name of the column that uniquely identifies the records in this table.
  - **$useAutoIncrement**: Specifies if the table uses an auto-increment feature for $primaryKey.
  - **$returnType**: This setting allows you to define the type of data that is returned.
  - **$allowedFields**: This array should be updated with the field names that can be set during save(), insert(), or update() methods. A

cidemo > app > Models > Customers.php

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class Customers extends Model
{
    protected $table            = 'customers';
    protected $primaryKey       = 'id';
    protected $useAutoIncrement = true;
    protected $returnType       = 'array';
    protected $useSoftDeletes   = false;
    protected $protectFields    = true;
    protected $allowedFields    = [];

}
```

How to transition to other web frameworks?

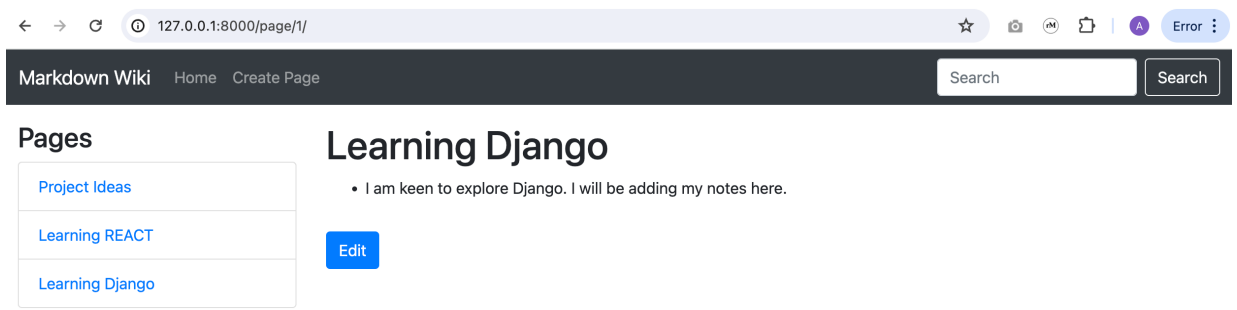
- Web fundamentals are the same.
  - HTML
  - Javascript
  - Forms (Post vs Get)
- MVC implementations are very similar
  - Eg Flask, Django, FastAPI
  - …. But there are differences
  - And different conventions and folder structures
  - And more powerful Models (including Object Relational Modeling) than the models in CodeIgniter
  - And different templating libraries (i.e. alternatives to embedded PHP in HTML)
  - And different ways to implement Routing
- More recently there are newer ideas
  - React (client-side)
  - Next.JS (React that runs on Server with new ideas for layouts)

Comparing Frameworks

- We will look at Django, React and Next.js

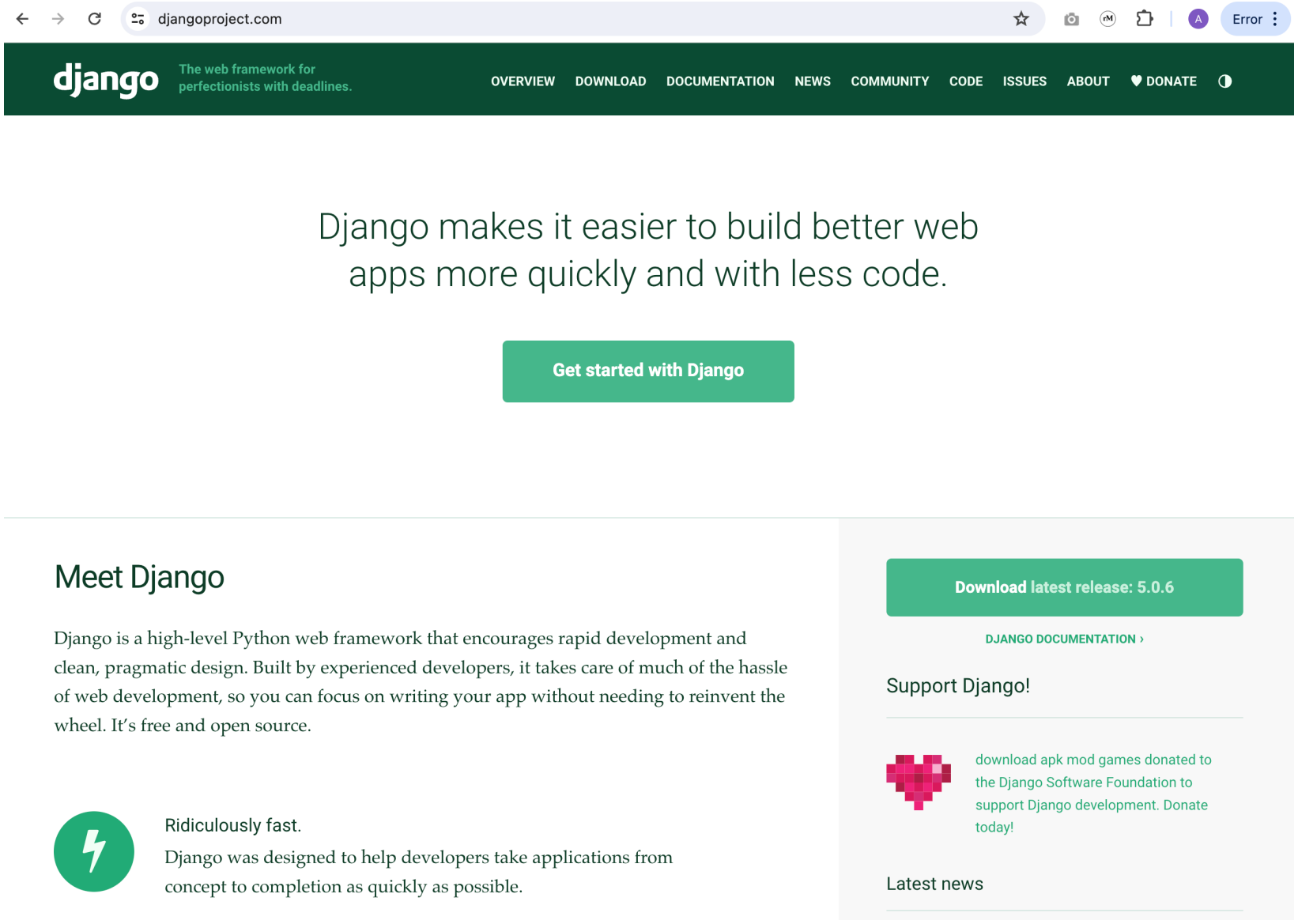
- We will cover:
  - Installation
  - Project Structure
  - Routing and Controllers
  - How to make a base template?
  - Models

- We will build a small WIKI app



Django
(Server-side Python Framework)

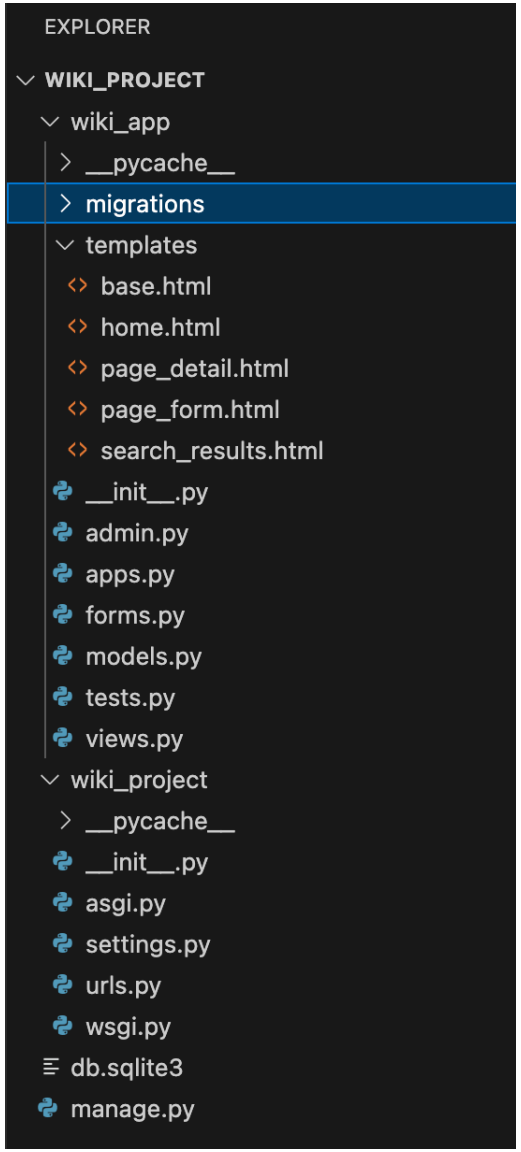
Django


djangoproject.com
Error
django
The web framework for perfectionists with deadlines.
OVERVIEW
DOWNLOAD
DOCUMENTATION
NEWS
COMMUNITY
CODE
ISSUES
ABOUT
DONATE
Django makes it easier to build better web apps more quickly and with less code.
Get started with Django
Meet Django
Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.
Ridiculously fast.
Django was designed to help developers take applications from concept to completion as quickly as possible.
Download latest release: 5.0.6
DJANGO DOCUMENTATION ›
Support Django!
download apk mod games donated to the Django Software Foundation to support Django development. Donate today!
Latest news

Creating a Django Project

- **Create a virtual environment**
  - `python3 -m venv venv`
  - `source venv/bin/activate`
- **Install Django and a Markdown library**
  - `pip install django`
  - `pip install markdown`
- **Create a Django project and app**
  - `django-admin startproject wiki_project`
  - `python manage.py startapp wiki_app`

Django Project Structure



The main directories are:

- wiki_app: Contains the application-specific code, including models, views, forms, and templates.
- wiki_project: Contains the project-level configuration files and settings.
  - settings.py: The main Django project settings file.
  - urls.py: The URL configuration file for the project.

Django Project Structure – Settings.py

wiki_project > settings.py

```python
"""
Django settings for wiki_project project.

Generated by 'django-admin startproject' using Django 5.0.6.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-fw7i$00a-kxh+5&)69utciul=4mij50jc5_jk5+g=2=$f2&yj4'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'wiki_app',  # Add this line
]
```

Django Project Structure – urls.py

wiki_project > urls.py

```python
"""
URL configuration for wiki_project project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/5.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""

from django.contrib import admin
from django.urls import path
from wiki_app.views import home, page_detail, search, page_form

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home, name='home'),
    path('page/<int:pk>/', page_detail, name='page_detail'),
    path('search/', search, name='search'),
    path('page/add/', page_form, name='add_page'),
    path('page/<int:pk>/edit/', page_form, name='edit_page')
]
```

Creating a Model for the WIKI

wiki_app > models.py

```python
from django.db import models

class Page(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

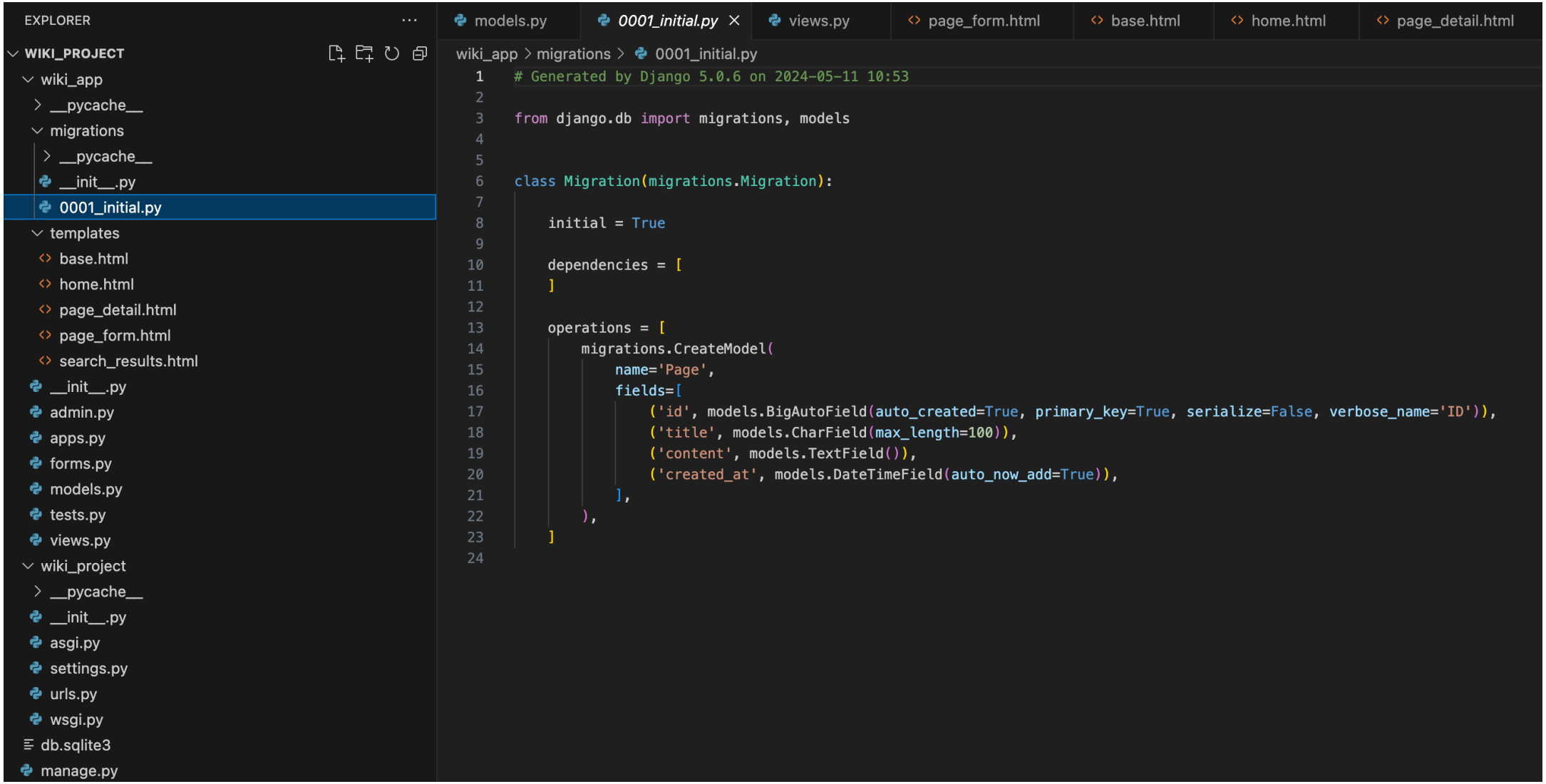
```
(venv) uqabakh1@uqabakh1-9893 wiki_project % python manage.py makemigrations
Migrations for 'wiki_app':
  wiki_app/migrations/0001_initial.py
    - Create model Page
(venv) uqabakh1@uqabakh1-9893 wiki_project % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, wiki_app
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying wiki_app.0001_initial... OK
```

In Django we don’t create the migration first.

We create a model using simple syntax. Then we generate the migration. Django can convert a model into the migration file.

This is much easier than CodeIgniter ☺

The Autogenerated Migration



Base Templates

wiki_app > templates > base.html > html > body > nav.navbar.navbar-expand-lg.navbar-dark.bg-dark > div#navbarNav.collapse.navbar-co

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Markdown Wiki</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <a class="navbar-brand" href="{% url 'home' %}">Markdown Wiki</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
                aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'home' %}">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'add_page' %}">Create Page</a>
                </li>
            </ul>
            <form class="form-inline my-2 my-lg-0" action="{% url 'search' %}" method="get">
                <input class="form-control mr-sm-2" type="search" name="q" placeholder="Search" aria-label="Search">
                <button class="btn btn-outline-light my-2 my-sm-0" type="submit">Search</button>
            </form>
        </div>
    </nav>

    <div class="container-fluid mt-3">
        <div class="row">
            <div class="col-md-3">
                <h3>Pages</h3>
                <ul class="list-group">
                    {% for page in pages %}
                        <li class="list-group-item">
                            <a href="{% url 'page_detail' page.pk %}">{{ page.title }}</a>
                        </li>
                    {% endfor %}
                </ul>
            </div>
            <div class="col-md-9">
                {% block content %}{% endblock %}
            </div>
        </div>
    </div>

```

- We can embed Python within HTML.
- This uses Jinja Templating
- We can also define blocks
- Blocks are like the CodeIgniter sections.

wiki_app > templates > home.html > a.btn.btn-primary.mt-3

```
{% extends 'base.html' %}

{% block content %}
    <h1>Welcome to Markdown Wiki</h1>

    <a href="{% url 'add_page' %}" class="btn btn-primary mt-3">Create New Page</a>
{% endblock %}
```

Views

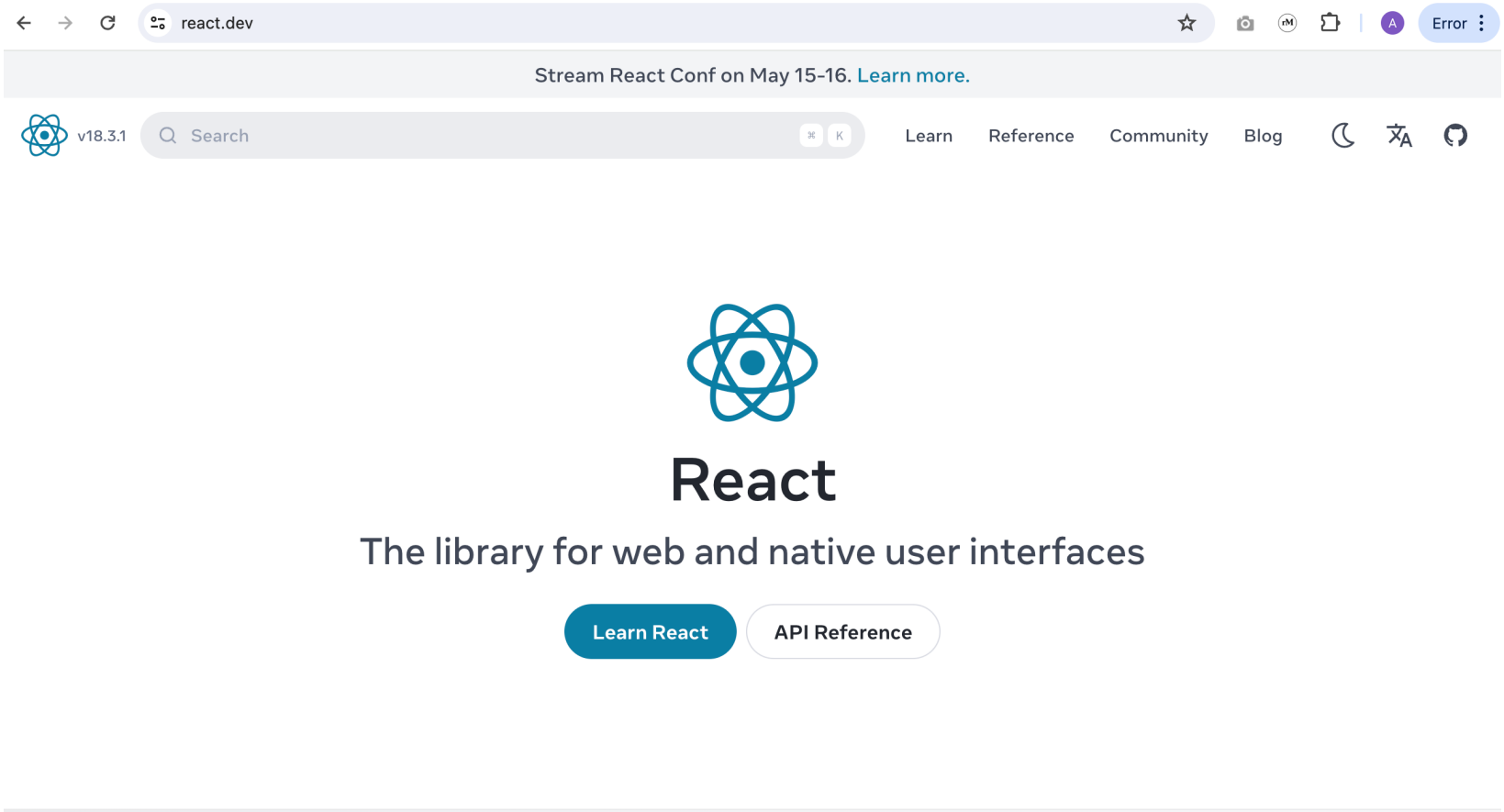
wiki_app > views.py

```python
from django.shortcuts import render, redirect, get_object_or_404
from .models import Page
from .forms import PageForm
import markdown

def get_pages():
    return Page.objects.order_by('-created_at')

def home(request):
    pages = get_pages()
    return render(request, 'home.html', {'pages': pages})

def page_detail(request, pk):
    page = get_object_or_404(Page, pk=pk)
    page.content = markdown.markdown(page.content)
    pages = get_pages()
    return render(request, 'page_detail.html', {'page': page, 'pages': pages})

def search(request):
    query = request.GET.get('q')
    pages = Page.objects.filter(title__icontains=query)
    all_pages = get_pages()
    return render(request, 'search_results.html', {'pages': pages, 'query': query, 'all_pages': all_pages})

def page_form(request, pk=None):
    if pk:
        page = get_object_or_404(Page, pk=pk)
    else:
        page = None

    if request.method == 'POST':
        form = PageForm(request.POST, instance=page)
        if form.is_valid():
            page = form.save()
            return redirect('page_detail', pk=page.pk)
    else:
        form = PageForm(instance=page)

    pages = get_pages()
    return render(request, 'page_form.html', {'form': form, 'pages': pages})
```

- The views.py file contains methods that can use models and render templates
- This is essentially the equivalent of the CodeIgniter Controllers
- Django Models are much more powerful. Django has a full Object Relational Mapper (ORM)
- Data is passed to the template

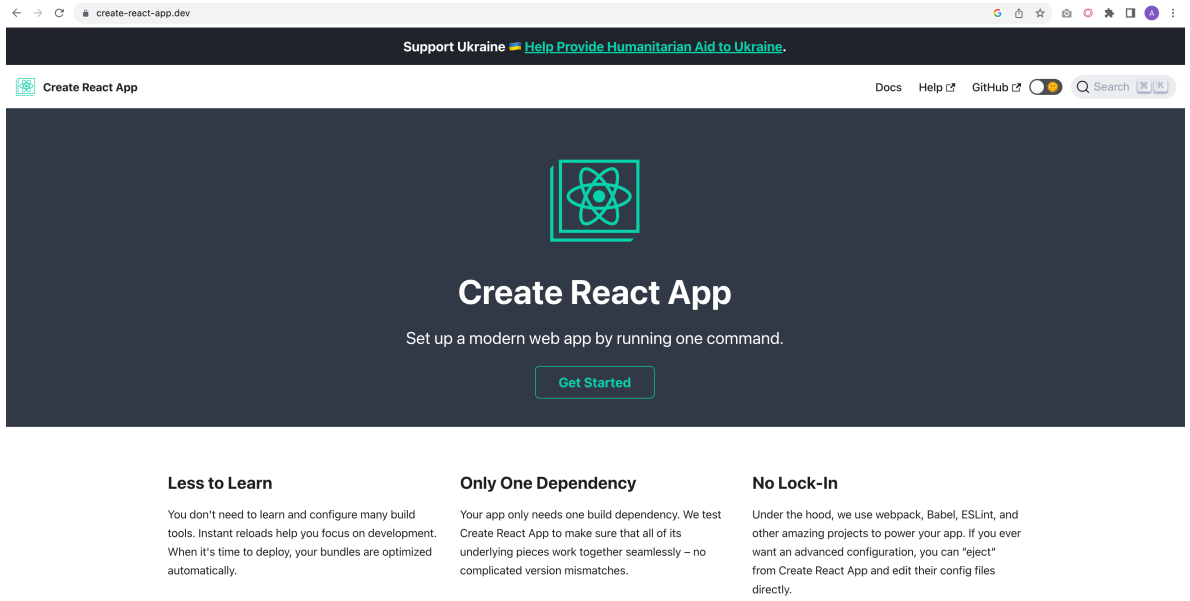
React
(Client-side Javascript Framework)

React


react.dev
Error
Stream React Conf on May 15-16. Learn more.
v18.3.1
Search
Learn
Reference
Community
Blog
React
The library for web and native user interfaces
Learn React
API Reference

Using create-react-app

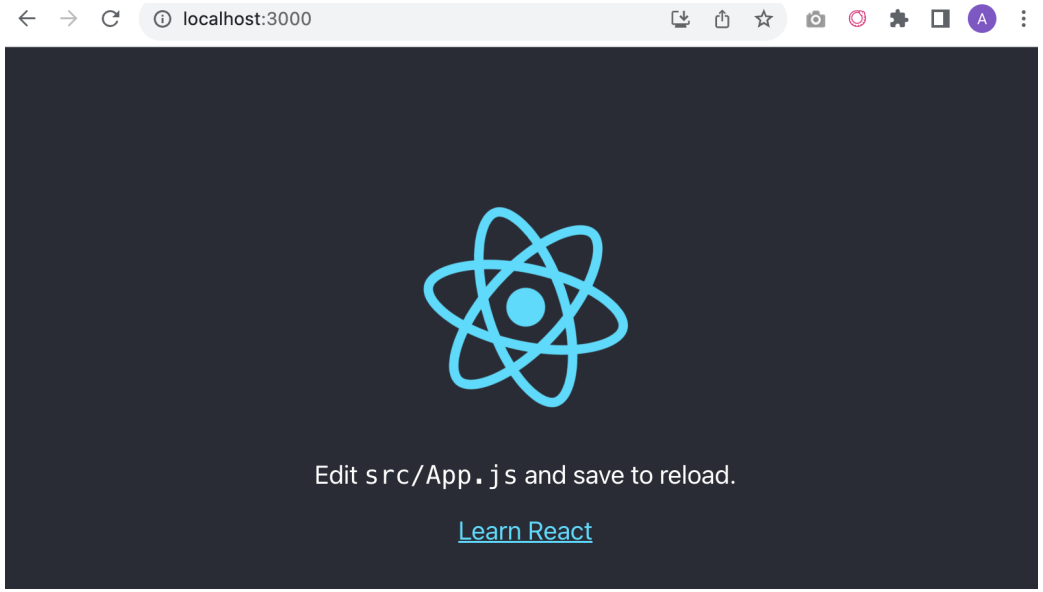
- **Create a new react client side application:**
  - npx create-react-app simple-app --use-npm
- **What does create-react-app do?**
  - A command-line tool developed by Meta (Facebook) that sets up a new React project with a recommended file structure and configuration.
  - It simplifies the process of starting a React application by automatically setting up a development environment, including the necessary dependencies and build scripts, without requiring manual configuration.



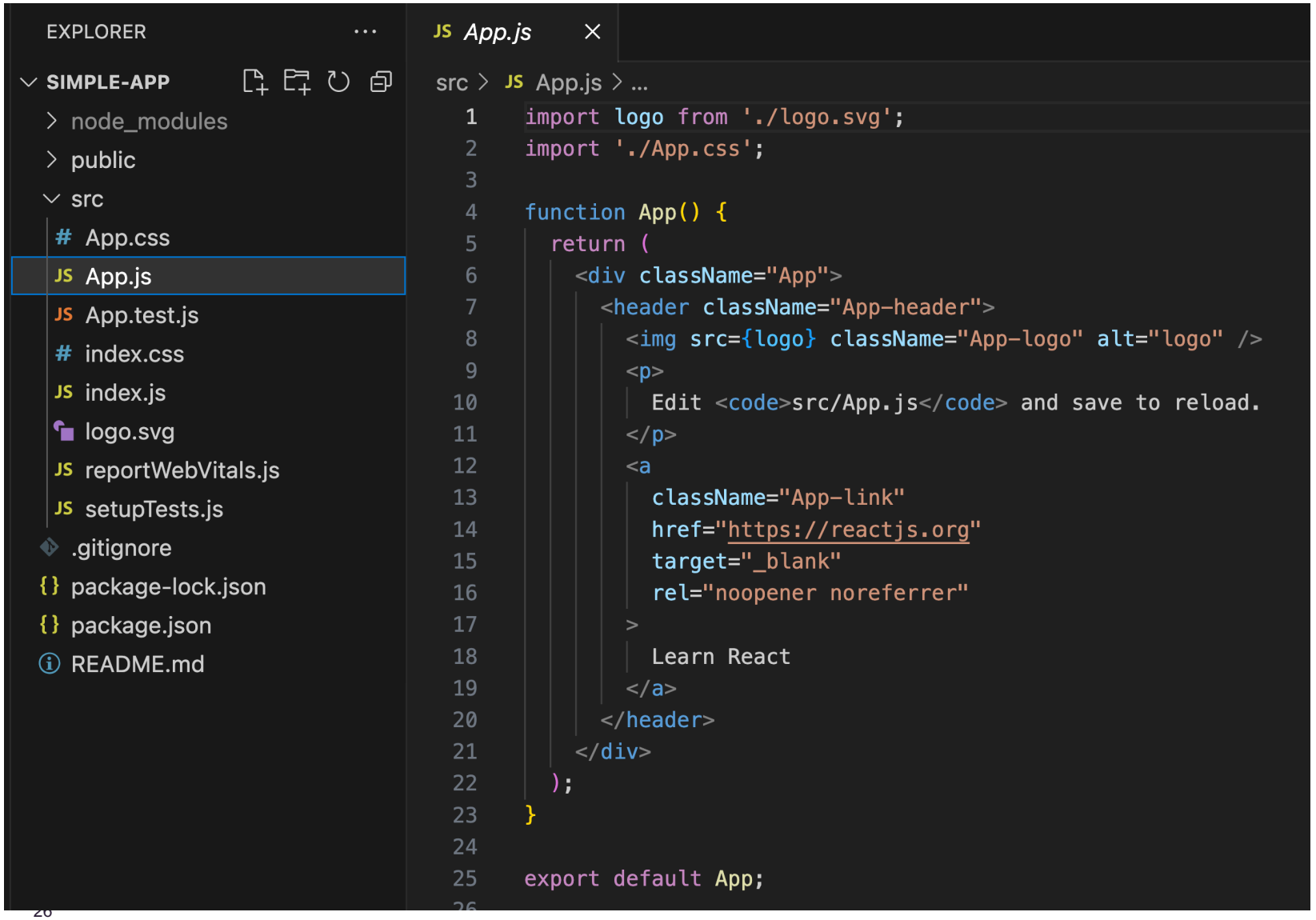
Creating a React Project

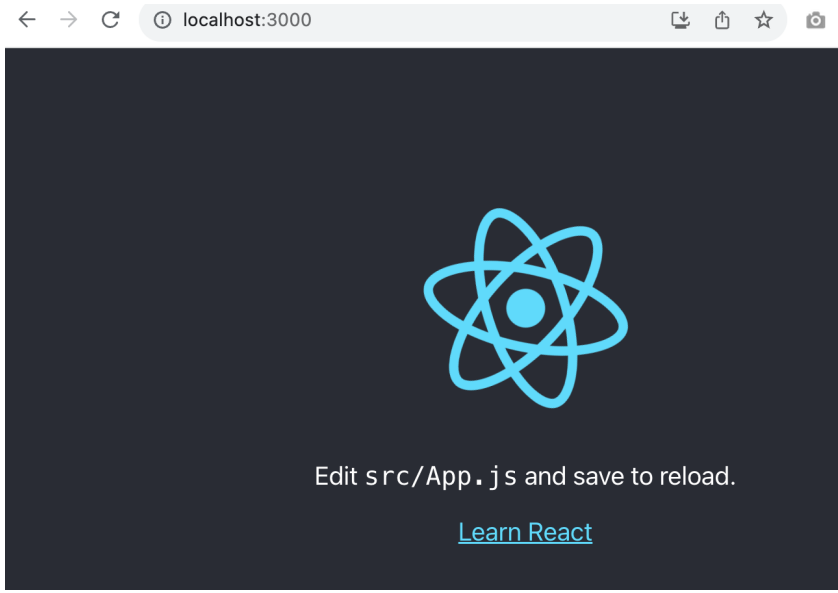
- **Create a new react client side application:**
  - npx create-react-app simple-app --use-npm
  - cd simple-app
  - npm start

```
Compiled successfully!

You can now view simple-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.0.0.113:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



Explore the created folders





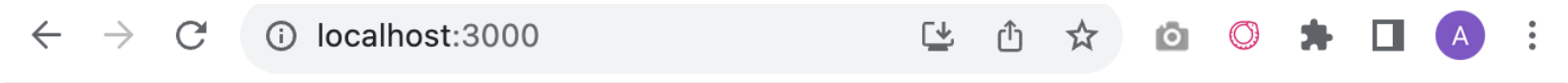
HelloWorld Component

src > components > JS HelloWorld.js > default

```
1  import React from 'react';
2
3  function HelloWorld() {
4    return (
5      <h1>Hello, World!</h1>
6    );
7  }
8
9  export default HelloWorld;
```

src > JS App.js > ...

```
1  import HelloWorld from './components/HelloWorld';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <HelloWorld />
8      </div>
9    );
10 }
11
12 export default App;
13
```



Hello, World!

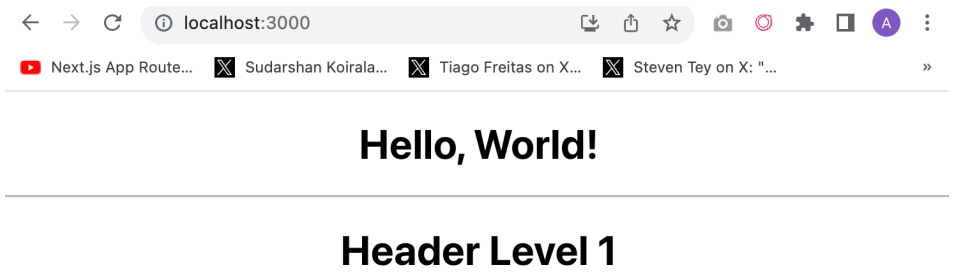
Header Component

src > components > JS Header.js > default

```
1  import React from 'react';
2
3  function Header(props) {
4    return (
5      <h1>{props.text}</h1>
6    );
7  }
8
9  export default Header;
```

src > JS App.js > ...

```
1  import HelloWorld from './components/HelloWorld';
2  import Header from './components/Header';
3  import './App.css';
4
5  function App() {
6    return (
7      <div className="App">
8        <HelloWorld />
9        <hr />
10       <Header text="Header Level 1" />
11     </div>
12   );
13 }
14
15 export default App;
16
```



ItemList Component

src > components > JS ItemList.js > ...

```
1   import React from 'react';
2
3   function ItemList() {
4     const items = ['Item 1', 'Item 2', 'Item 3'];
5
6     return (
7       <ul>
8         {items.map((item, index) => (
9           <li key={index}>{item}</li>
10        ))}
11      </ul>
12    );
13  }
14
15  export default ItemList;
16
```

Hello, World!

Header Level 1

- Item 1
- Item 2
- Item 3

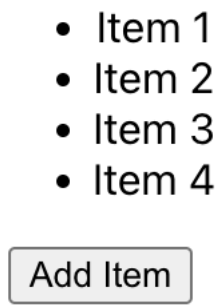
src > JS App.js > ...

```
1   import HelloWorld from './components/HelloWorld';
2   import Header from './components/Header';
3   import ItemList from './components/ItemList';
4   import './App.css';
5
6   function App() {
7     return (
8       <div className="App">
9         <HelloWorld />
10        <hr />
11        <Header text="Header Level 1" />
12        <hr />
13        <ItemList />
14      </div>
15    );
16  }
17
18  export default App;
19
```

Add Item with a Button

src > components > JS ItemListWithState.js > ...

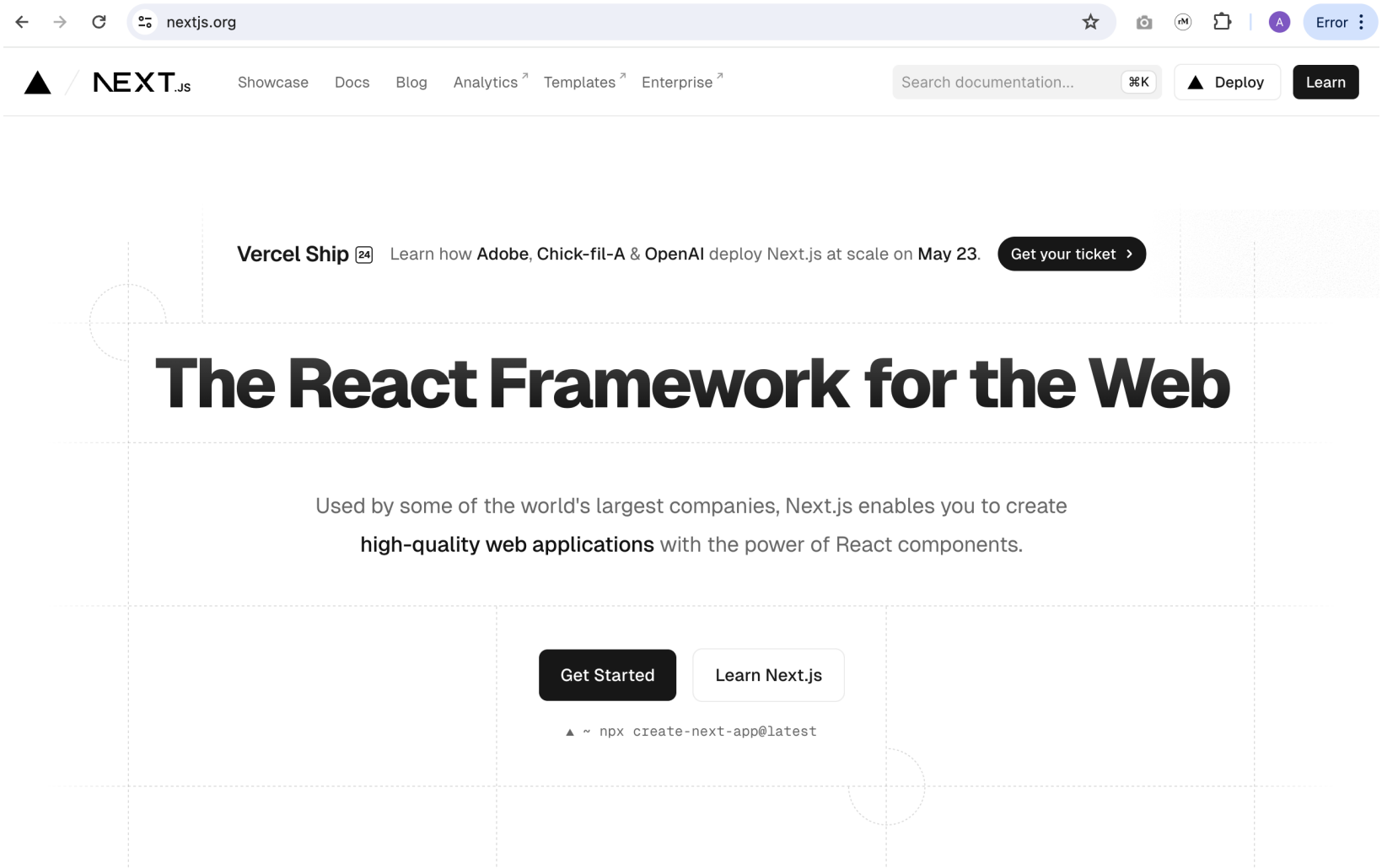
```
1   import React, { useState } from 'react';
2
3   function ItemListWithState() {
4     const [items, setItems] = useState(['Item 1', 'Item 2', 'Item 3']);
5
6     const addItem = () => {
7       const newItem = `Item ${items.length + 1}`;
8       setItems([...items, newItem]);
9     };
10
11    return (
12      <div style={{ marginTop: '20px', display: 'grid', placeItems: 'center' }}>
13        <ul>
14          {items.map((item, index) => (
15            <li key={index}>{item}</li>
16          ))}
17        </ul>
18        <button onClick={addItem}>Add Item</button>
19      </div>
20    );
21  }
22
23  export default ItemListWithState;
24
```



- **What have we not had to program?**
- **State is a very powerful concept!**
- **When state is updated the component is re-rendered**

Next.JS

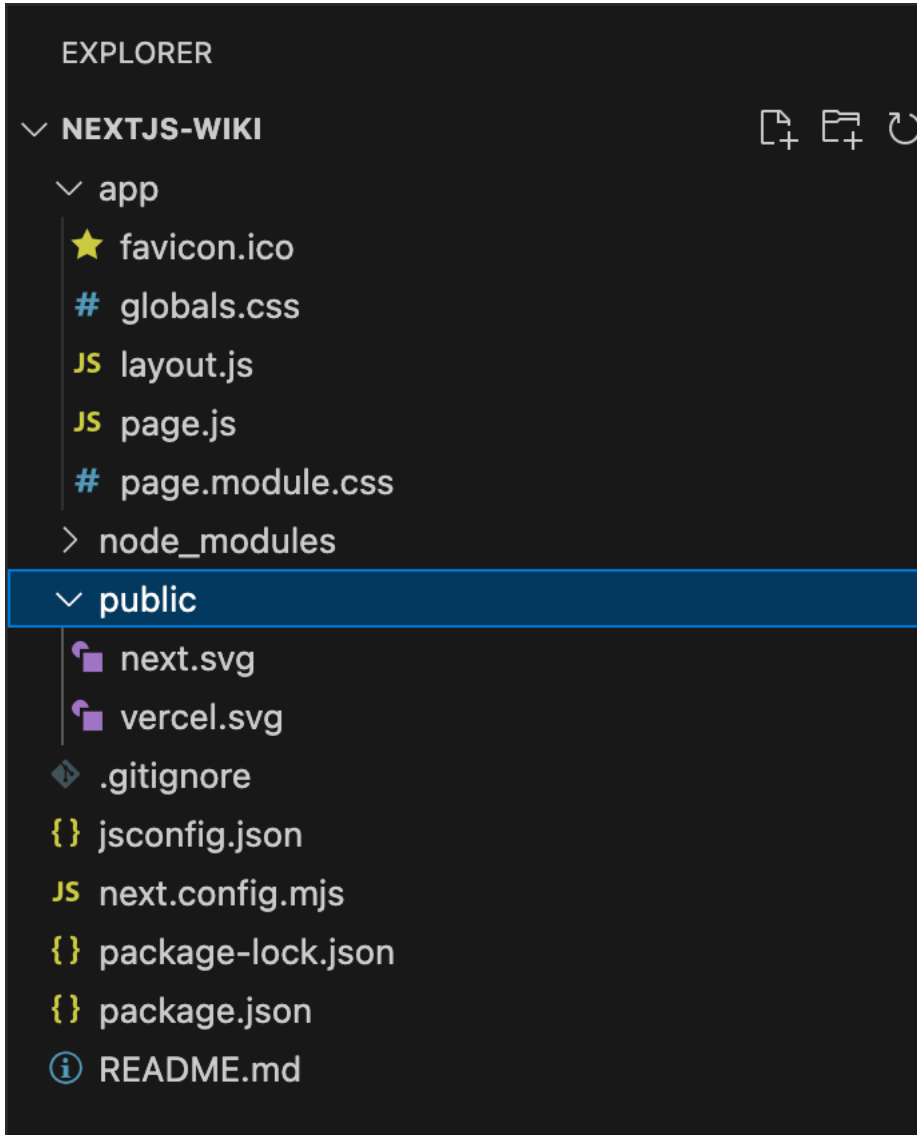
Next.js


nextjs.org
Error
NEXT.JS
Showcase
Docs
Blog
Analytics
Templates
Enterprise
Search documentation...
⌘K
Deploy
Learn
Vercel Ship 24 Learn how Adobe, Chick-fil-A & OpenAI deploy Next.js at scale on May 23.
Get your ticket
The React Framework for the Web
Used by some of the world's largest companies, Next.js enables you to create
high-quality web applications with the power of React components.
Get Started
Learn Next.js
▲ ~ npx create-next-app@latest

Using create-next-app

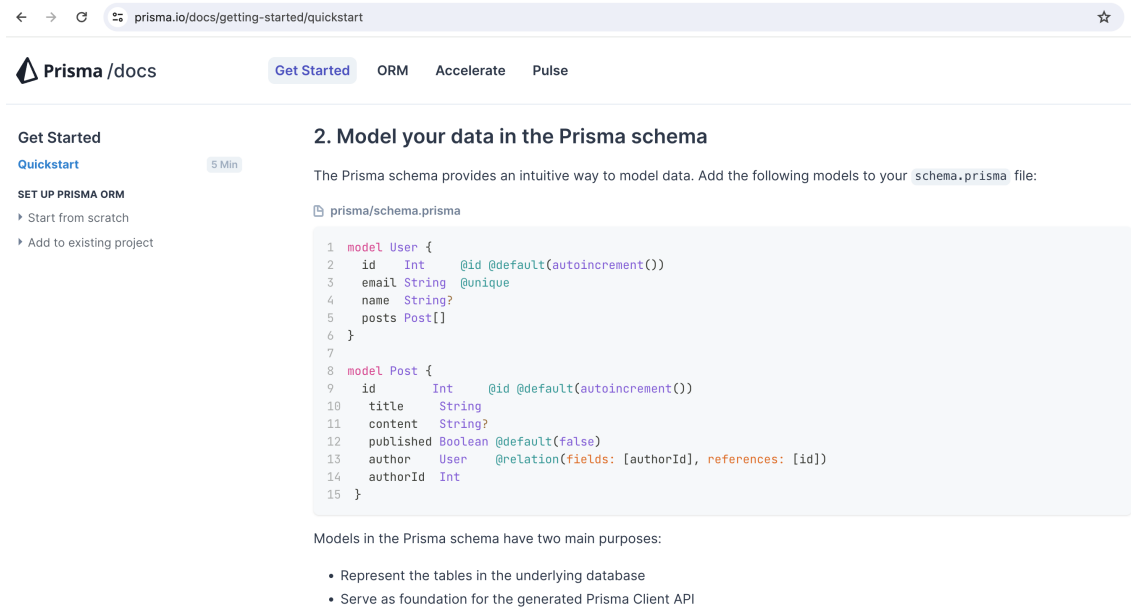
- **Install Node.js** https://nodejs.org/en
- **Create a new react client side application:**
  - `npx create-next-app@latest`

```
(venv) uqabakh1@uqabakh1-9893 nextjswiki % npx create-next-app@latest
Need to install the following packages:
create-next-app@14.2.3
Ok to proceed? (y) y
✔ What is your project named? … nextjs-wiki
✔ Would you like to use TypeScript? … No / Yes
✔ Would you like to use ESLint? … No / Yes
✔ Would you like to use Tailwind CSS? … No / Yes
✔ Would you like to use `src/` directory? … No / Yes
✔ Would you like to use App Router? (recommended) … No / Yes
✔ Would you like to customize the default import alias (@/*)? … No / Yes
Creating a new Next.js app in /Users/uqabakh1/INFS3202/lectures/week12/week12_code/nextjswiki/nextjs-wiki.

Using npm.

Initializing project with template: app


Installing dependencies:
- react
- react-dom
- next


added 21 packages, and audited 22 packages in 15s

3 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Success! Created nextjs-wiki at /Users/uqabakh1/INFS3202/lectures/week12/week12_code/nextjswiki/nextjs-wiki
```



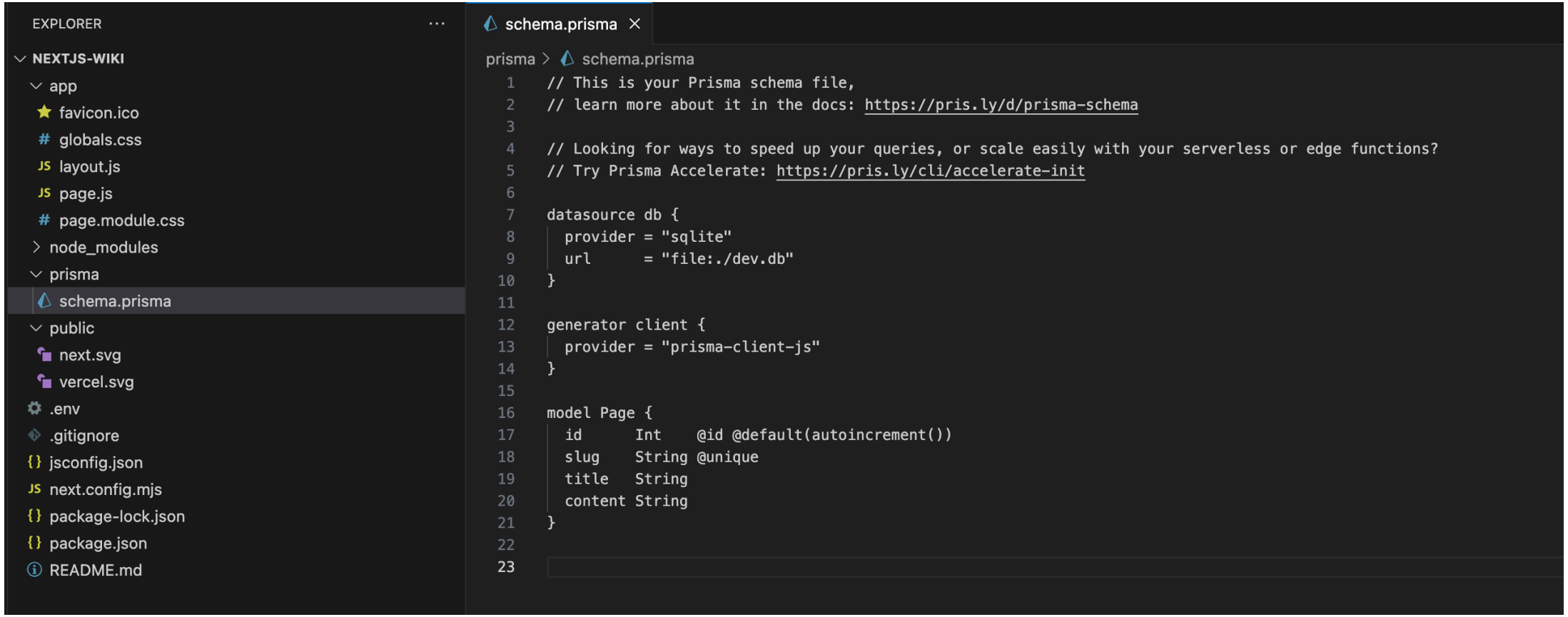
Install Dependencies

- **We will use Prisma as our ORM and Bootstrap for CSS**
  - `npm install @prisma/client bootstrap`
- **Initialise Prisma**
  - npx prisma init




```
(venv) uqabakh1@uqabakh1-9893 nextjs-wiki % npx prisma init


✔ Your Prisma schema was created at prisma/schema.prisma
  You can now open it in your favorite editor.

warn You already have a .gitignore file. Don't forget to add `.env` in it to not commit any private information.

Next steps:
1. Set the DATABASE_URL in the .env file to point to your existing database. If your database has no tables yet, read https://pris.ly/d/getting-started
2. Set the provider of the datasource block in schema.prisma to match your database: postgresql, mysql, sqlite, sqlserver, mongodb or cockroachdb.
3. Run prisma db pull to turn your database schema into a Prisma schema.
4. Run prisma generate to generate the Prisma Client. You can then start querying your database.

More information in our documentation:
https://pris.ly/d/getting-started

  Developing real-time features?
  Prisma Pulse lets you respond instantly to database changes.
  https://pris.ly/cli/pulse
```

Create the schema (i.e. Model)



Generate the Prisma Client

- **Run this command to create the client**
  - `npx prisma generate`

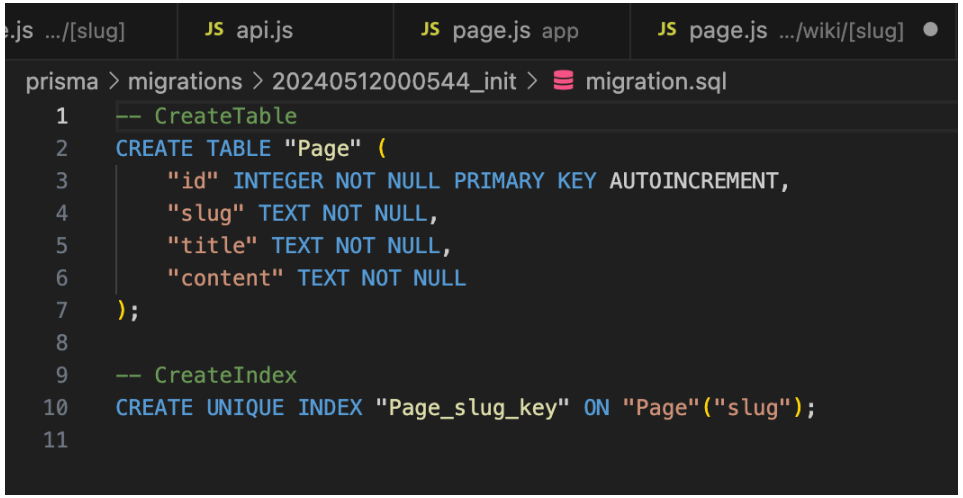
Prisma Client is an auto-generated and type-safe query builder for Node.js

- **Auto-generated:** Prisma Client is automatically generated based on your Prisma schema file (schema.prisma). It provides an intuitive and type-safe API for querying and modifying your database.
- **Query builder:** Prisma Client provides a fluent and expressive query builder API. It allows you to construct complex database queries using a chain of methods, making your code more readable and maintainable.
- **CRUD operations:** Prisma Client supports all the common CRUD (Create, Read, Update, Delete) operations. It provides methods like create, findUnique, findMany, update, delete, etc., to perform these operations on your database entities.

```
uqabakh1@uqabakh1-9893 nextjs-wiki % npx prisma generate
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma

✔ Generated Prisma Client (v5.13.0) to ./node_modules/@prisma/client in 49ms

Start using Prisma Client in Node.js (See: https://pris.ly/d/client)
```
```
import { PrismaClient } from '@prisma/client'
const prisma = new PrismaClient()
```
```
or start using Prisma Client at the edge (See: https://pris.ly/d/accelerate)
```
```
import { PrismaClient } from '@prisma/client/edge'
const prisma = new PrismaClient()
```
```
See other ways of importing Prisma Client: http://pris.ly/d/importing-client

  Supercharge your Prisma Client with global database caching,
  scalable connection pooling and real-time database events.
  Explore Prisma Accelerate: https://pris.ly/cli/-accelerate
  Explore Prisma Pulse: https://pris.ly/cli/-pulse
```

Create the database table (similar to Migrations)

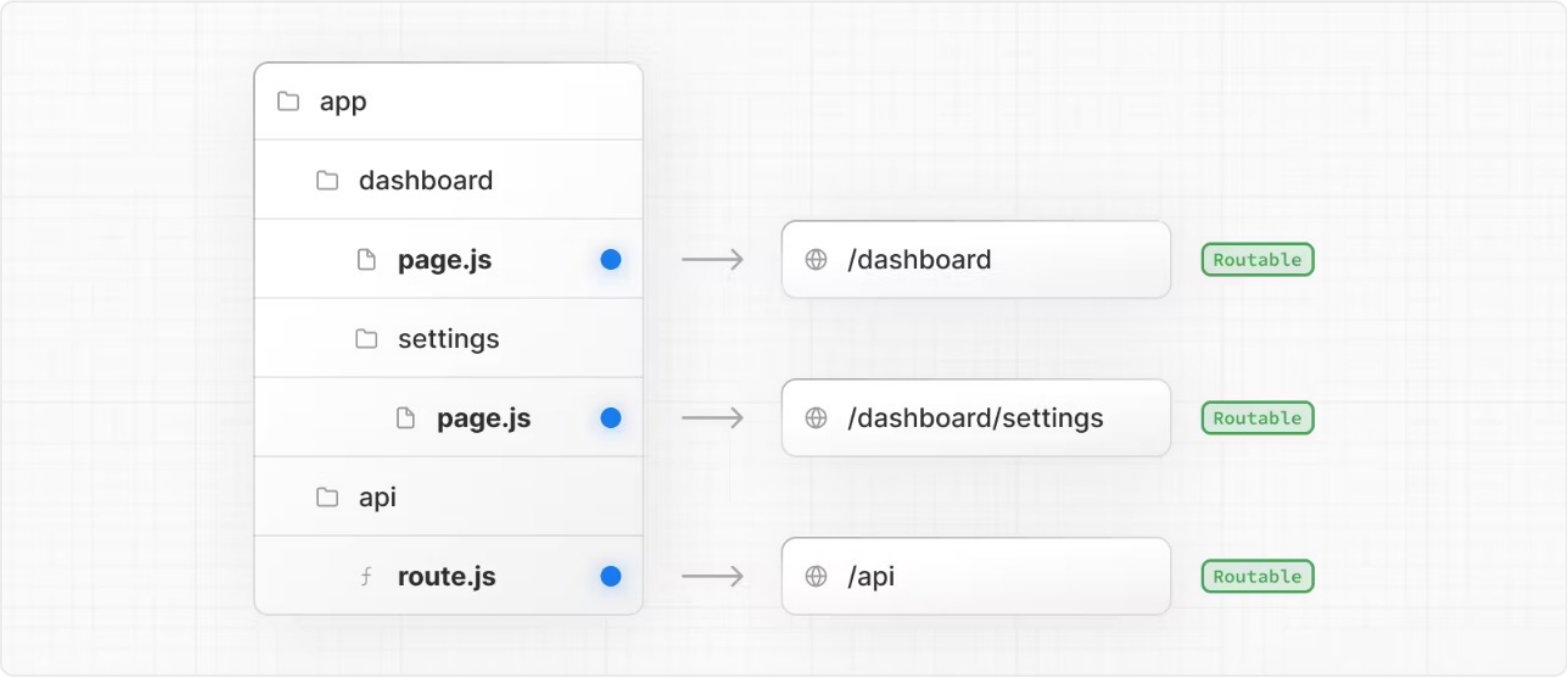
- **Run this command to create the client**
  - `npx prisma migrate dev --name init`




```
uqabakh1@uqabakh1-9893 nextjs-wiki % npx prisma migrate dev --name init
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": SQLite database "dev.db" at "file:./dev.db"

SQLite database dev.db created at file:./dev.db

Applying migration `20240512000544_init`

The following migration(s) have been created and applied from new schema changes:

migrations/
  └─ 20240512000544_init/
    └─ migration.sql

Your database is now in sync with your schema.

✔ Generated Prisma Client (v5.13.0) to ./node_modules/@prisma/client in 43ms
```

Page based Routing

- **Include a folder within app and it becomes the route**
- **Must have page.js within the folder**



Dynamic Page based Routing

A Dynamic Segment can be created by wrapping a folder's name in square brackets: `[folderName]`. For example, `[id]` or `[slug]`.

Dynamic Segments are passed as the `params` prop to `layout`, `page`, `route`, and `generateMetadata` functions.

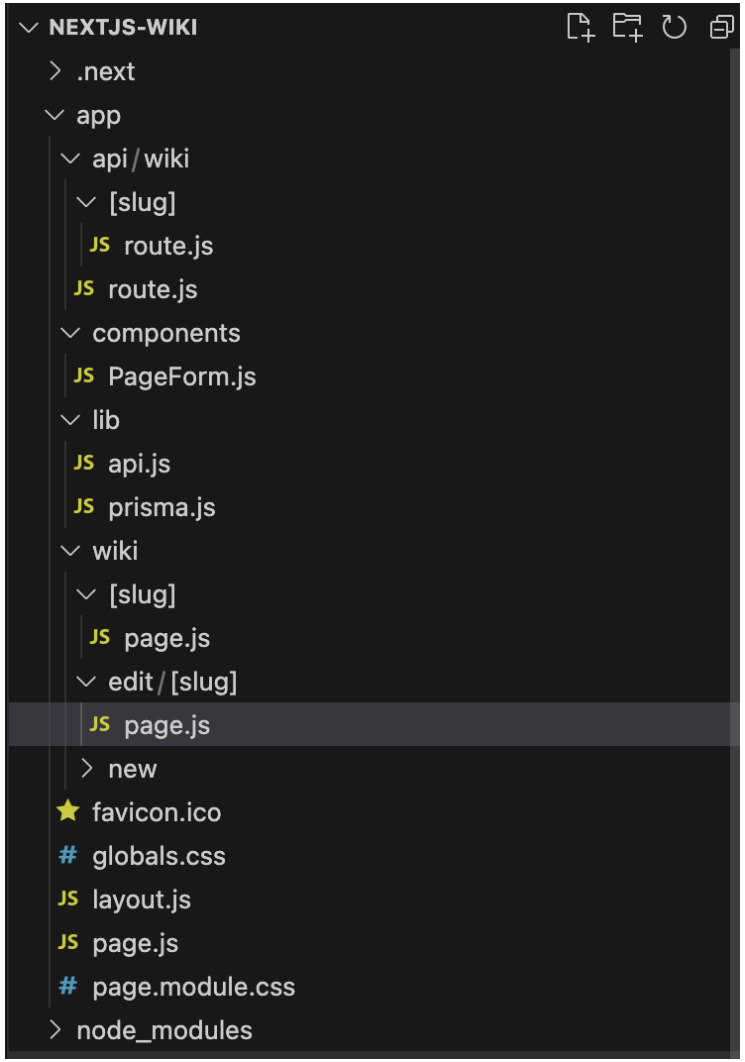
Example

For example, a blog could include the following route `app/blog/[slug]/page.js` where `[slug]` is the Dynamic Segment for blog posts.

app/blog/[slug]/page.tsx — TypeScript

```tsx
export default function Page({ params }: { params: { slug: string } }) {
  return <div>My Post: {params.slug}</div>
}
```

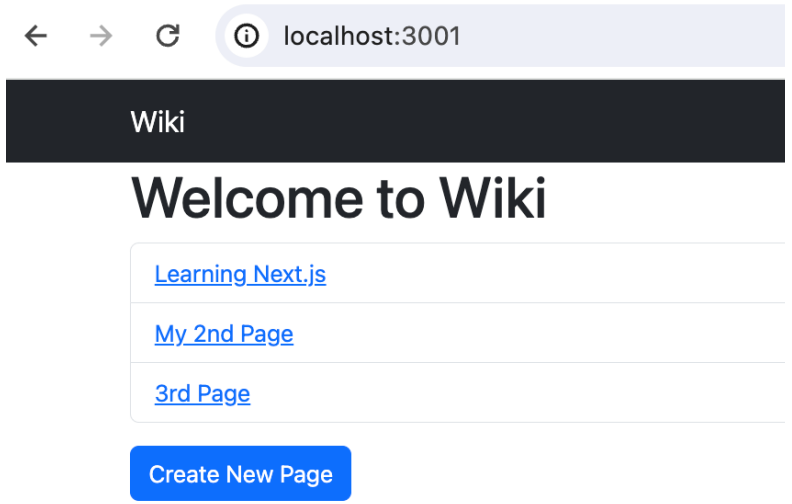
Route	Example URL	`params`
`app/blog/[slug]/page.js`	`/blog/a`	`{ slug: 'a' }`
`app/blog/[slug]/page.js`	`/blog/b`	`{ slug: 'b' }`
`app/blog/[slug]/page.js`	`/blog/c`	`{ slug: 'c' }`



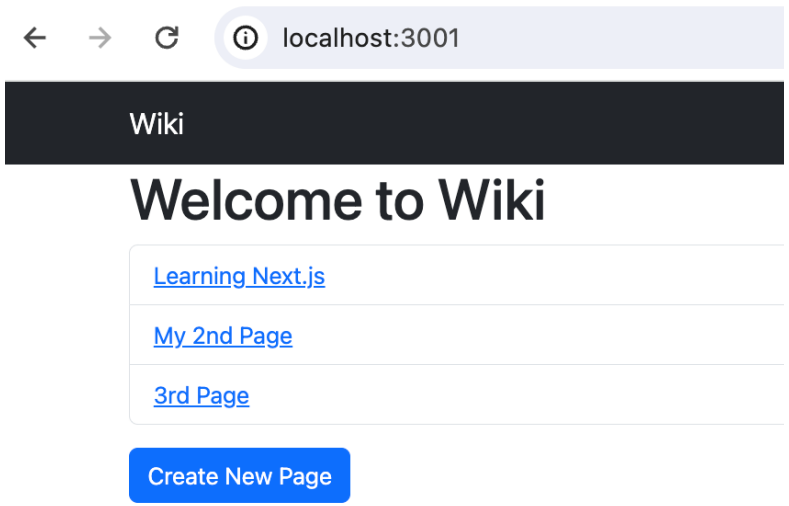
Run Next.js Dev Server

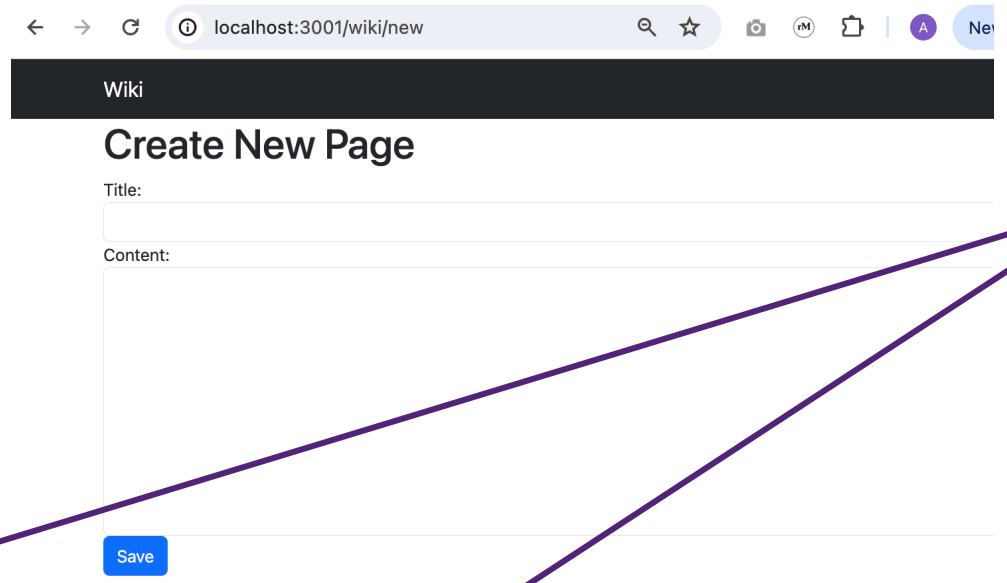
- **Run the dev server**
  - `npm run dev`

Should run on port 3000 unless that is being used. It will then increment the port number eg 3001

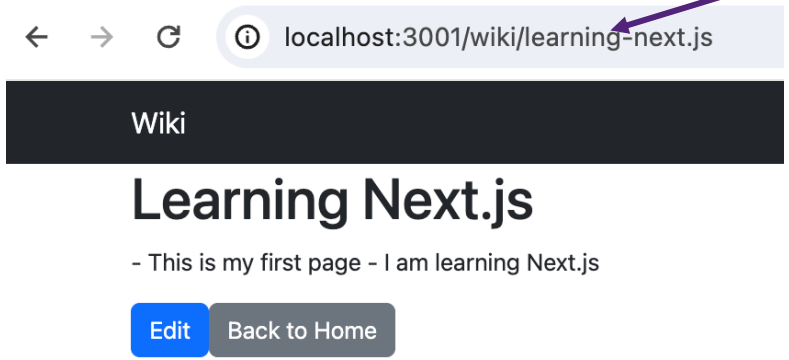


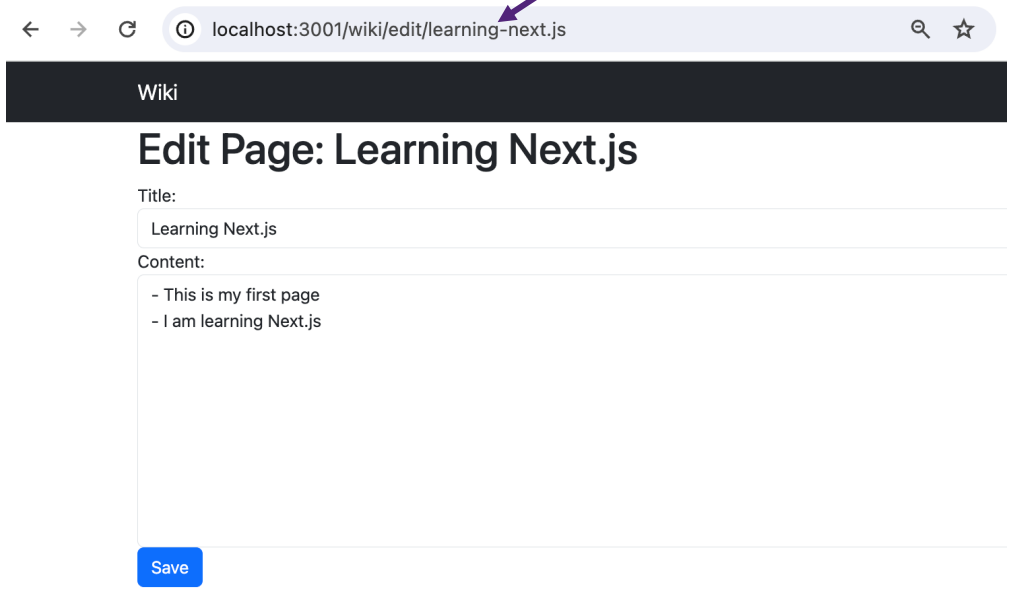
Wiki Routes in Action





Dynamic route
`[slug]`





Layout

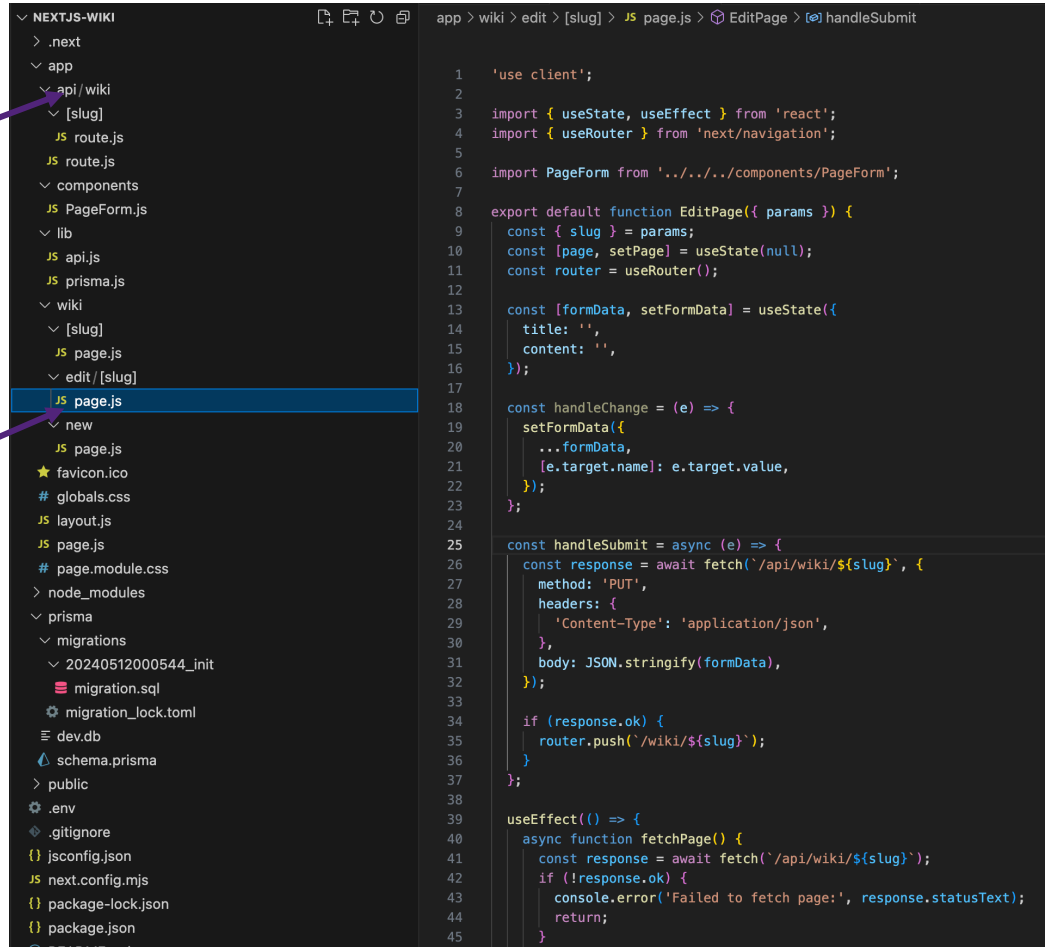
app > JS layout.js > RootLayout

```
1  import 'bootstrap/dist/css/bootstrap.css';
2  import Link from 'next/link';
3
4  export default function RootLayout({ children }) {
5    return (
6      <html lang="en">
7        <body>
8          <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
9            <div className="container">
10             <Link href="/" className="navbar-brand">
11               Wiki
12             </Link>
13           </div>
14         </nav>
15         <main>{children}</main>
16       </body>
17     </html>
18   );
19 }
```

Client-side & Server-side Code

Api runs on server
By default js and React code runs on server

Include ‘use client’ to run within Web Browser



Example API call:

Other Frameworks

Flask

flask.palletsprojects.com/en/3.0.x/

Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Chat

Contents

Quick search

Go

Flask

Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs describe each component of Flask in detail, with a full reference in the API section.

Flask depends on the Werkzeug WSGI toolkit, the Jinja template engine, and the Click CLI toolkit. Be sure to check their documentation as well as Flask's when looking for information.

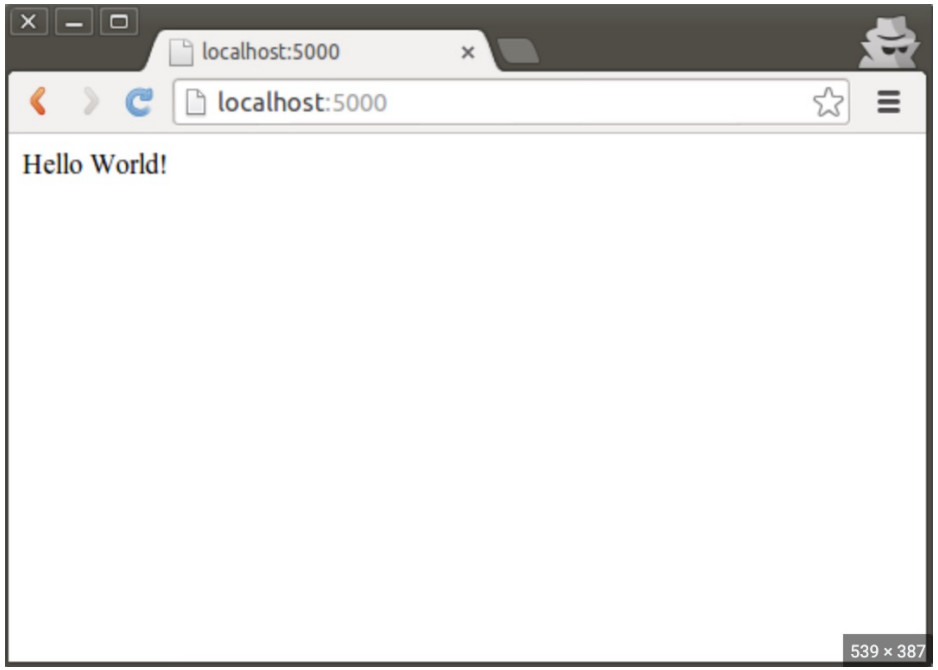
User's Guide

Flask provides configuration and conventions, with sensible defaults, to get started. This section of the documentation explains the different parts of the Flask framework and how they can be used, customized, and extended. Beyond Flask itself, look for community-maintained extensions to add even more functionality.

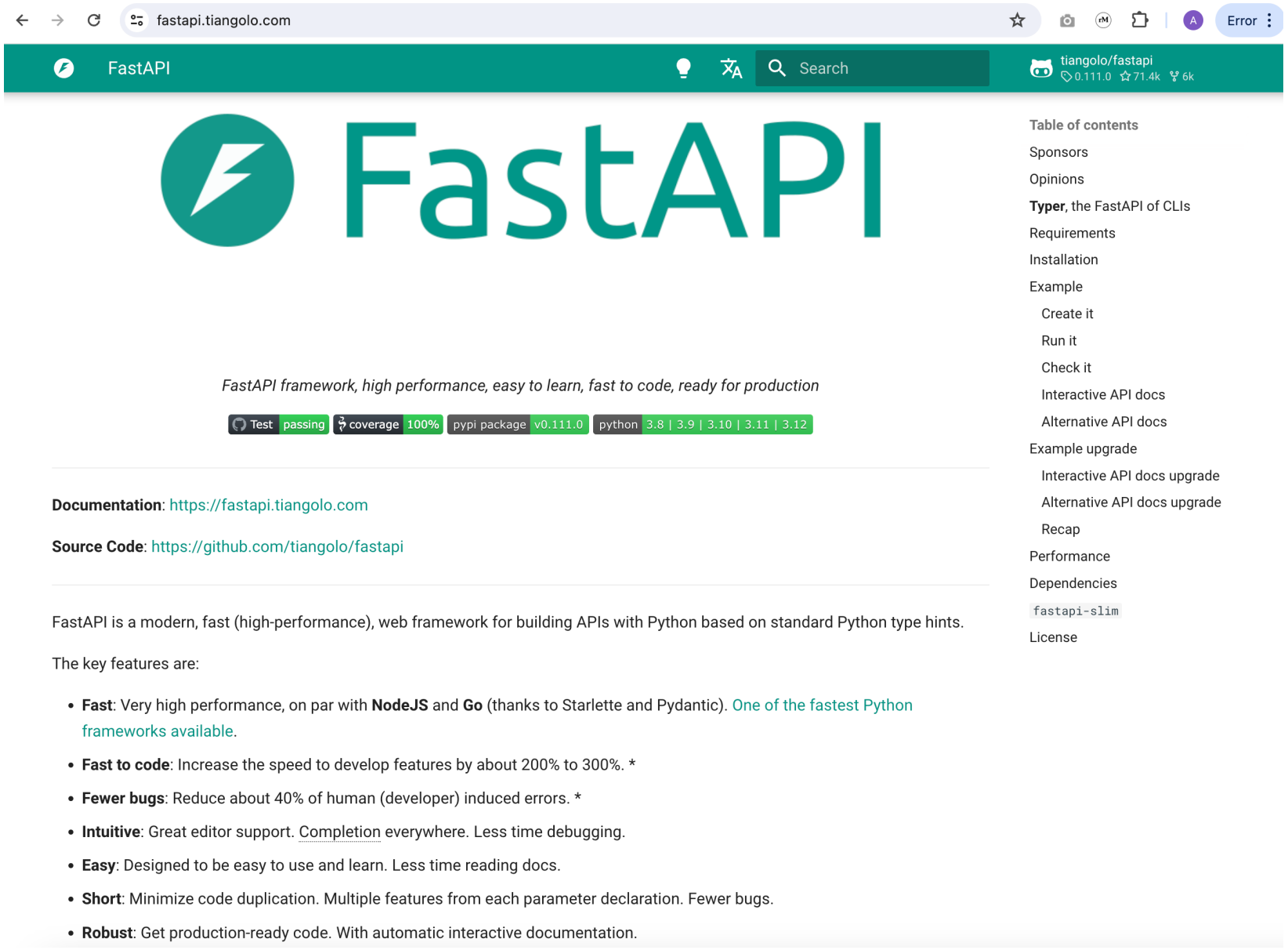
Flask

- Very concise syntax
- Uses decorators in Python to define Routes
- Need to install your own ORM eg SQL Alchemy

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello() -> str:
    return "Hello World"


if __name__ == "__main__":
    app.run(debug=False)
```



FastAPI


fastapi.tiangolo.com
Error
FastAPI
Search
tiangolo/fastapi
0.111.0 71.4k 6k
FastAPI
FastAPI framework, high performance, easy to learn, fast to code, ready for production
Test passing
coverage 100%
pypi package v0.111.0
python 3.8 | 3.9 | 3.10 | 3.11 | 3.12
Documentation: https://fastapi.tiangolo.com
Source Code: https://github.com/tiangolo/fastapi
FastAPI is a modern, fast (high-performance), web framework for building APIs with Python based on standard Python type hints.
The key features are:
Fast: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic). One of the fastest Python frameworks available.
Fast to code: Increase the speed to develop features by about 200% to 300%. *
Fewer bugs: Reduce about 40% of human (developer) induced errors. *
Intuitive: Great editor support. Completion everywhere. Less time debugging.
Easy: Designed to be easy to use and learn. Less time reading docs.
Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
Robust: Get production-ready code. With automatic interactive documentation.
Table of contents
Sponsors
Opinions
Typer, the FastAPI of CLIs
Requirements
Installation
Example
Create it
Run it
Check it
Interactive API docs
Alternative API docs
Example upgrade
Interactive API docs upgrade
Alternative API docs upgrade
Recap
Performance
Dependencies
fastapi-slim
License

SECaTs for Semester 1, 2024

13 May – 31 May (11:59 pm AEST)

What are the benefits of completing the survey?

The SECaT is your opportunity to provide feedback on your learning experience!

Your responses will be used to improve courses and teaching for future students. Please be assured that all responses are strictly confidential.

How do I complete the survey?

Visit https://eval.uq.edu.au/eus.onlinesurveyportal/ or scan this QR code:

Please provide specific examples where possible. Respectful constructive feedback is always helpful to help improve your learning experience.

Week 12: Todo

- Continue with your Project Assessment Item
- Project Assessment Item is due Friday 17 May at 3pm
- In the Lab you can work on your project
- If you don’t have a Student Card, please get one!
  - You will need a Student Card for the Exam

Thank you