Developing AI Applications with Python and Flask

Maulana Hafidz Ismail

16 Oktober 2025

Contents

1 Python Coding Practices and Packaging Concepts

1.1 Introduction

Python dengan Flask adalah framework aplikasi web yang ringan dan fleksibel. Framework ini dikenal karena kesederhanaan, minimalis, dan kemudahan penggunaannya. Dirancang sebagai micro-framework yang menyediakan struktur ringan yang memudahkan pengembang membangun aplikasi web dengan cepat dan mudah, tanpa mengorbankan efisiensi dan kemampuan untuk meningkatkan skala proyek dari skala kecil ke aplikasi yang lebih besar dan kompleks.

1.2 Python with Flask for large-scale development

Flask adalah pilihan yang baik untuk aplikasi yang lebih kecil dan sederhana. Namun, istilah `"mikro"` lebih berkaitan dengan apa itu Flask, alih-alih membatasi potensi skalabilitasnya. Flask dapat digunakan untuk sistem berskala besar dan aplikasi yang lebih kompleks dengan memperhatikan persyaratan dan batasan spesifik, perencanaan yang cermat, arsitektur yang baik, dan desain modular. Namun, Flask mungkin memerlukan upaya pengelolaan dan penskalaan yang lebih besar dibandingkan dengan kerangka kerja yang lebih tangguh dan kaya fitur.

Ekosistemnya yang kaya dan tangguh menyediakan alat, pustaka, dan fungsionalitas bagi pengembang untuk menangani tugas-tugas pengembangan web seperti perutean, penanganan permintaan, rendering templat, atau tugas serupa. Caching, penyeimbangan beban, replikasi, dan penyimpanan data Anda secara skalabel dapat membantu mencapai hasil yang optimal.

1.3 Key Flask Capabilities

- **Extensibility and integration**: Flask dapat diperluas dan pengembang dapat menambahkan atau menghapus fitur yang memungkinkan kustomisasi. Flask terintegrasi secara mulus dengan pustaka dan kerangka kerja Python lainnya, memungkinkan pengembang untuk menggabungkan fungsionalitasnya dengan alat dan teknologi lain, sehingga meningkatkan kemampuannya.

- **Transparent documentation**: Dokumentasi Flask diterbitkan, memungkinkan pengembang untuk menggunakan API dan utilitas internalnya serta menemukan titik kait, penggantian, dan sinyal sesuai kebutuhan.

- **Custom implementation**: Kustomisasi bawaan dan kelas khusus dapat digunakan untuk hal-hal seperti objek permintaan dan respons. Kelas Flask memiliki banyak metode yang dirancang untuk subkelas. Anda dapat dengan cepat menambahkan atau menyesuaikan perilaku dengan membuat subkelas Flask dan menggunakan subkelas tersebut di mana pun Anda membuat instance kelas aplikasi.

- **Scaling considerations**: Anda dapat menggunakan penskalaan sedemikian rupa sehingga jika Anda menggandakan jumlah server, Anda akan mendapatkan kinerja sekitar dua kali lipat. Hanya ada satu faktor pembatas terkait penskalaan di Flask, yaitu penggunaan proksi lokal konteks. Proksi ini bergantung pada konteks yang dalam Flask didefinisikan sebagai thread, proses, atau greenlet. Jika server Anda menggunakan jenis konkurensi yang tidak berbasis thread atau greenlet, Flask tidak akan lagi dapat mendukung proksi global ini.

- **Modular development**: Carilah cara-cara agar proyek Anda dapat direfaktor menjadi kumpulan utilitas dan ekstensi Flask. Jelajahi berbagai ekstensi di komunitas dan cari pola untuk membangun ekstensi Anda sendiri jika Anda tidak menemukan alat yang dibutuhkan. Cara terbaik untuk meningkatkan alat untuk aplikasi yang lebih besar adalah dengan mendapatkan umpan balik dari pengguna.

1.4 Real-world applications

Saat ini, Python dengan Flask telah menjadi pilihan populer di kalangan perusahaan besar karena kesederhanaan, fleksibilitas, fleksibilitas, serta kemudahan pembelajaran dan penggunaannya. Desainnya yang minimalis dan sifatnya yang dapat dikustomisasi menjadikannya adaptif, efektif, dan andal untuk kebutuhan pengembangan web skala besar di berbagai industri dan sektor.

Beberapa perusahaan terkemuka, termasuk Netflix, Reddit, Lyft, LinkedIn, Pinterest, dan Uber, memanfaatkan Python dengan Flask dalam tumpukan teknologi mereka untuk layanan atau fungsionalitas backend tertentu. Python Flask menguntungkan perusahaan besar untuk berbagai tujuan seperti pengembangan API, layanan backend, pengembangan cepat, dan pembuatan prototipe, sementara ekstensibilitasnya memfasilitasi penambahan fungsionalitas ke dalam infrastruktur mereka. Hal ini menunjukkan bahwa Python Flask dapat menjadi bagian dari arsitektur yang skalabel jika dikombinasikan dengan strategi dan alat yang tepat.

1.5 Fase dalam Application Development Lifecycle

- Requirement Gathering:

  Mengumpulkan kebutuhan dari pengguna, bisnis, dan teknis, pantangan (constraint). Contoh: Pengguna harus dapat melihat berbagai kamar dan fasilitas yang tersedia.

- Analysis:

  Menganalisis kebutuhan yang telah dikumpulkan untuk merancang solusi yang mungkin. Penting untuk mendokumentasikan semua pembaruan dalam desain.

- Design:

Merancang solusi lengkap berdasarkan analisis yang dilakukan. Dokumentasi yang jelas dan ringkas diperlukan untuk fase berikutnya.

- Code and Test:

  Menggunakan dokumentasi desain untuk mengkode, menguji, dan merevisi aplikasi hingga memenuhi semua spesifikasi. Unit Testing: Pengujian pada level unit untuk memastikan setiap bagian kode berfungsi sesuai harapan.

- User and System Test:

  Pengujian dari sudut pandang pengguna dan pengujian sistem untuk memastikan aplikasi berfungsi dengan baik. Integration Testing: Memastikan semua program berfungsi setelah diintegrasikan. Performance Testing: Mengukur kecepatan, skalabilitas, dan stabilitas aplikasi.

- Production:

  Aplikasi tersedia untuk pengguna akhir. Aplikasi harus dalam keadaan stabil dan tidak mengalami perubahan besar.

- Maintenance:

  Aplikasi mungkin memerlukan pembaruan atau penambahan fitur baru. Fitur baru harus melalui semua fase sebelumnya sebelum diintegrasikan ke dalam aplikasi yang sudah ada.

1.6 Praktik Terbaik dalam Pengkodean

- Menggunakan beberapa file untuk mengkode berbagai fungsionalitas.
- Membuat program pusat yang memanggil file-file individual untuk menjalankan fungsi tertentu.

1.7 Web Applications

- Definisi: Web application adalah program yang disimpan di server jarak jauh dan diakses melalui internet menggunakan browser. Contoh: situs e-commerce, webmail.
- Komponen: Terdapat tiga komponen utama untuk memproses permintaan klien:
  - Web Server: Mengelola permintaan yang diajukan.
  - App Server: Menjalankan tugas yang diminta.
  - Database: Menyimpan informasi yang diperlukan untuk menyelesaikan tugas.
- Pengembangan: Kode ditulis untuk front-end menggunakan JavaScript, HTML, atau CSS, dan untuk back-end menggunakan Python, Java, atau Ruby.

1.8 Keuntungan Web Applications

- Versi aplikasi yang sama dapat diakses oleh banyak pengguna secara bersamaan.
- Pengguna dapat menggunakan aplikasi dari berbagai platform (desktop, laptop, mobile).
- Aplikasi dapat diakses melalui browser tanpa perlu instalasi.

1.9 Application Programming Interface (API)

- Definisi: API adalah komponen perangkat lunak yang memungkinkan dua aplikasi yang tidak terhubung untuk berkomunikasi. API memiliki aturan dan fungsi standar untuk menentukan data yang dapat diambil atau dimodifikasi.
- Contoh: Aplikasi cuaca yang meminta data dari weather API untuk memberikan ramalan cuaca.
- Arsitektur: API dapat dibangun menggunakan arsitektur seperti **Representational State Transfer (REST)** atau **Simple Object Access Protocol (SOAP)**.
- Manfaat API:
  - Meningkatkan konektivitas antara aplikasi.
  - Mendukung tindakan CRUD (Create, Read, Update, Delete).
  - Bekerja dengan HTTP verbs seperti PUT, POST, DELETE, dan GET.
  - Dapat disesuaikan karena berbasis HTTP.

1.10 Perbandingan Web Application dan APIs

- Semua web applications adalah APIs, tetapi tidak semua APIs adalah web applications.
- Contoh: Dalam layanan belanja e-commerce, browser bertindak sebagai API yang menghubungkan pengguna dengan web application.

1.11 PEP-8 Guidelines untuk Keterbacaan Kode

- Indentasi: Gunakan empat spasi untuk setiap level indentasi. Ini penting karena editor teks dapat menginterpretasikan tab dengan cara yang berbeda, yang dapat menyebabkan kesalahan format.
- Baris Kosong: Gunakan baris kosong untuk memisahkan fungsi dan kelas. Ini membantu dalam memahami struktur kode dengan lebih baik.

1.12 Konvensi Pengkodean untuk Konsistensi

- Fungsi Terpisah: Buat fungsi terpisah untuk blok kode yang lebih besar. Ini meningkatkan kecepatan eksekusi dan mendukung penggunaan kembali blok kode. Contoh sintaks:

```
def function_one(a, b):
return a + b
```

- Penamaan Fungsi dan File: Gunakan huruf kecil dengan garis bawah (lowercase with underscores) untuk nama fungsi dan file. Ini mengikuti konvensi Python yang umum. Contoh: `def calculate_area()`
- Penamaan Kelas: Gunakan CamelCase untuk nama kelas. Ini membantu membedakan antara kelas dan fungsi. Contoh: `class MyClass`
- Konstanta: Gunakan huruf kapital dengan garis bawah untuk nama konstanta. Ini menunjukkan tujuan konstanta tersebut. Contoh: `MAX_FILE_UPLOAD_SIZE`

1.13 Analisis Kode Statis

Analisis kode statis, atau analisis statis, adalah aktivitas verifikasi efisiensi kode aplikasi yang menganalisis kode sumber untuk kualitas, keandalan, dan keamanan tanpa mengeksekusi kode tersebut. Analisis kode statis merupakan bagian penting dari setiap siklus pengembangan aplikasi dan tersedia sebagai bagian dari berbagai kerangka kerja dengan Python.

Salah satu kerangka kerja yang paling populer adalah PyLint. PyLint pada dasarnya mengevaluasi kode berdasarkan kepatuhan terhadap panduan gaya pengkodean PEP8 dan menghasilkan komentar setiap kali menemukan masalah.

- Static Code Analysis: Metode untuk mengevaluasi kode terhadap gaya dan standar yang telah ditentukan tanpa mengeksekusi kode. Ini membantu menemukan masalah seperti kesalahan pemrograman dan pelanggaran standar pengkodean Contoh alat: **PyLint** digunakan untuk memeriksa kepatuhan kode Python terhadap pedoman PEP-8.

1.14 Definisi Unit Testing

- Unit Testing: Metode untuk memvalidasi apakah unit kode (bagian kecil dari aplikasi) berfungsi dengan baik.

1.15 Proses Unit Testing

- Pengujian Dua Fase: Local Testing: Pengujian unit dilakukan di sistem lokal. Jika gagal, penyebabnya harus diidentifikasi dan diperbaiki.

- Server Testing:

  Setelah lulus pengujian lokal, unit diuji di server (misalnya, CICD server). Jika gagal, detail kegagalan akan diberikan untuk perbaikan.

1.16 Membangun dan Menjalankan Unit Tests

- Unit Test Library:

  Digunakan untuk membuat unit tests. Ini adalah modul Python yang menyediakan kerangka kerja untuk pengujian.

- Penamaan File:

  File unit test harus memiliki awalan atau akhiran dengan kata `"test"` untuk membedakannya dari file unit.

1.17 Langkah-langkah Membuat File Unit Test

- Import Library:

```
import unittest
```

  unittest: Library Python untuk membuat dan menjalankan unit tests.

- Import Fungsi yang Diuji:

```
from mymodule import square, doubler
```

- Membangun Kelas Unit Testing:

```
class TestMyModule(unittest.TestCase):
```

  TestMyModule: Kelas yang digunakan untuk mengelompokkan unit tests. Harus mewarisi dari unittest.TestCase.

- Membuat Fungsi Test:

```
def test_square(self):
self.assertEqual(square(2), 4)
```

  assertEqual: Metode untuk membandingkan dua nilai. Jika nilai tidak sama, test akan gagal.

1.18 Evaluasi Hasil Test

- Setelah menjalankan file test, output akan menunjukkan hasil test. Contoh output:
  - Jika test berhasil: OK
  - Jika test gagal: Menampilkan detail kegagalan, seperti:

```
Fail: test_square(_main_.TestMyModule)
AssertionError: 8 is not equal to 4
```

1.19 Kesimpulan terkait Unit Test

Unit Testing adalah metode penting untuk memastikan bahwa setiap unit kode berfungsi dengan baik. Proses ini melibatkan pengujian di lingkungan lokal dan server, serta penggunaan library unittest untuk membangun dan menjalankan test. Output dari unit test memberikan informasi yang berguna untuk memperbaiki kesalahan sebelum aplikasi diterapkan.

1.20 Module

- Definisi: Module adalah file `.py` yang berisi definisi Python, pernyataan, fungsi, dan kelas yang dapat diimpor ke dalam skrip lain.
- Contoh Sintaks:

```
def square(number):
return number ** 2
```

  Fungsi square ini mengembalikan kuadrat dari input.

1.21 Package

- Definisi: Package adalah kumpulan dari beberapa modules dalam sebuah direktori yang memiliki file `__init__.py`, yang membedakannya dari direktori biasa.
- Contoh Struktur:

```
my_project/
__init__.py
module1.py
module2.py
```

  File `__init__.py` digunakan untuk menginisialisasi package.

1.22 Library

- Definisi: Library adalah kumpulan dari packages atau bisa juga berupa satu package. Contoh library termasuk NumPy, PyTorch, dan Pandas. Catatan: Istilah package dan library sering digunakan secara bergantian.
- Contoh Struktur Library:

```
my_library/
__init__.py
package1/
    __init__.py
    module1.py
    module2.py
package2/
    __init__.py
    module3.py
    module4.py
package3/
    __init__.py
    module5.py
```

- Pengunaan:

```
from my_library.package1 import module1
```

1.23 Langkah-langkah Membuat Package

- Buat folder dengan nama package.
- Buat file kosong `__init__.py`.
- Buat modules yang diperlukan.
- Tambahkan kode dalam `__init__.py` untuk merujuk ke modules yang diperlukan. Contoh kode dalam `__init__.py`:

```
from . import module1
from . import module2
```

1.24 Verifikasi Package

Verifikasi dilakukan untuk memastikan package dapat diimpor tanpa error.
Langkah-langkah:

- Buka terminal bash.
- Pastikan direktori sama dengan folder package.
- Jalankan Python dengan perintah python.
- Ketik `import my_project` untuk memverifikasi.

1.25 Menggunakan Package

- Definisi: Setelah package dibuat, Anda dapat menggunakannya dalam skrip lain jika folder package berada dalam direktori yang sama. Contoh Sintaks:

```
    from my_project.module1 import square, doubler
    from my_project.module2 import mean

    print(square(4))  # Output: 16
    print(doubler(4))  # Output: 8
    print(mean(2, 1, 3))  # Output: 2
```

1.26

-

2 Web App Deployment using Flask

2.1 Python Libraries

- Definisi: Python libraries adalah kumpulan alat yang menyediakan fungsi tertentu untuk mempermudah dan mempercepat tugas pemrograman.
- Contoh Libraries:
  1. NumPy: Memfasilitasi perhitungan matematis yang kompleks.
  2. Pandas: Menawarkan kemampuan manipulasi dan analisis data.
  3. Matplotlib: Mempermudah visualisasi data.
  4. Requests: Menyederhanakan pengiriman HTTP requests.
  5. BeautifulSoup: Memudahkan pengambilan informasi dari halaman web.
  6. SQLAlchemy: Toolkit SQL dan alat Object-Relational Mapping (ORM) untuk manajemen database.
  7. PyTest: Framework pengujian yang memungkinkan pembuatan tes sederhana hingga kompleks.

2.2 Frameworks

- Definisi: Framework adalah struktur yang telah ditentukan untuk pengembangan aplikasi, menyediakan pedoman dan struktur untuk menulis dan mengorganisir kode.
- Contoh Frameworks:
  1. Django
  2. Flask
  3. Web2Py

2.3 Manfaat Menggunakan Framework

- Menyederhanakan Proses Pengembangan: Dengan kode yang sudah ditulis sebelumnya, pengembang dapat menghemat waktu.
- Mempermudah Debugging: Framework menyediakan alat debugging yang sudah dibangun.
- Fungsionalitas Lebih dengan Kode yang Lebih Sedikit: Pengembang dapat memanfaatkan pustaka dan modul yang sudah ada.
- Efisiensi Manajemen Database: Framework dilengkapi dengan alat integrasi database.
- Keamanan: Memanfaatkan fitur keamanan yang sudah ada dalam framework.

2.4 Perbedaan antara Libraries dan Frameworks

- Libraries: Kumpulan paket yang melakukan fungsi tertentu.
- Framework: Menyediakan alur dasar dan arsitektur aplikasi, memungkinkan pembangunan aplikasi secara keseluruhan.

2.5 Definisi Flask

Flash adalah sebuah micro framework untuk membuat aplikasi web dengan minimal dependencies. Flask tidak mengikat pengguna pada satu set alat tertentu.

2.6 Instalasi Flask

- Untuk menginstal Flask, disarankan untuk membuat virtual environment menggunakan modul venv atau bin venv.
- Sintaks untuk menginstal Flask:

```
    pip install Flask==2.2.2
```

2.7 Fitur Utama Flask

- Web Server: Menjalankan aplikasi dalam mode pengembangan.
- Debugger: Membantu dalam debugging aplikasi dengan menampilkan traceback interaktif di browser.
- Logging: Menggunakan logging standar Python untuk mencatat log aplikasi.
- Testing: Mendukung pengujian bagian-bagian aplikasi dengan framework seperti pytest dan coverage.

2.8 Fitur Tambahan Flask

- Static Assets: Mendukung file statis seperti CSS, JavaScript, dan gambar.
- Jinja Templating: Menggunakan Jinja untuk membuat halaman dinamis.
- Routing: Mendukung URL dinamis dan pengalihan untuk layanan RESTful.
- Error Handling: Menyediakan penanganan kesalahan global di tingkat aplikasi.
- Session Management: Mendukung manajemen sesi pengguna.

2.9 Ekstensi Komunitas Populer

- Flask-SQLAlchemy:

  Menambahkan dukungan untuk Object Relational Mapping (ORM) menggunakan SQLAlchemy, memungkinkan pengembang untuk bekerja dengan objek database dalam Python.

- Flask-Mail:

  Memungkinkan pengaturan server email SMTP untuk mengirim email dari aplikasi Flask.

- Flask-Admin:

  Memudahkan penambahan antarmuka admin ke aplikasi Flask, memungkinkan pengelolaan data dengan lebih mudah.

- Flask-Uploads:

  Menyediakan kemampuan untuk menambahkan pengunggahan file yang disesuaikan ke aplikasi Anda.

- Flask-CORS:

  Memungkinkan aplikasi Anda untuk menangani Cross-Origin Resource Sharing, sehingga permintaan JavaScript lintas domain dapat dilakukan.

- Flask-Migrate:

  Menambahkan dukungan untuk migrasi database menggunakan SQLAlchemy ORM, memudahkan pengelolaan perubahan skema database.

- Flask-User:

  Menyediakan manajemen pengguna, termasuk otentikasi dan otorisasi, serta fitur manajemen pengguna lainnya.

- Marshmallow:

  Menyediakan dukungan untuk serialisasi dan deserialisasi objek yang lebih luas, memudahkan konversi data antara format yang berbeda.

- Celery: Merupakan antrian tugas yang kuat untuk menjalankan tugas latar belakang sederhana hingga program multi-storage yang kompleks dan terjadwal.

2.10 Perbandingan dengan Django

- Flask: Framework ringan dengan fleksibilitas tinggi, hanya menyediakan dependencies dasar.

- Django: Framework full-stack yang menyediakan semua yang diperlukan untuk membuat aplikasi lengkap, tetapi lebih opinionated.

2.11 Membuat Aplikasi Flask

- Instalasi Flask: Pastikan Flask sudah terinstal sebelum membuat aplikasi.
- File Server: Buat file Python yang akan menjadi server, misalnya `app.py`.
- Instansiasi Flask:

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

- Dengan menggunakan `__name__`, Flask dapat mengetahui lokasi file yang sedang dijalankan, sehingga dapat menemukan file-file yang diperlukan untuk aplikasi Anda, seperti template HTML dan file statis (CSS, JavaScript, gambar, dll.).

2.12 Menambahkan Rute Pertama

Gunakan dekorator `(@app)` untuk mendefinisikan rute.

```
@app.route("/")
def hello_world():
    return "<b>my first Flask application in action!</b>"
```

Route adalah jalur URL yang mengarah ke fungsi tertentu dalam aplikasi.

2.13 Menjalankan Aplikasi

Set variabel `FLASK_APP` dan `FLASK_ENV` sebelum menjalankan aplikasi.

```
    export FLASK_APP=app.py
    export FLASK_ENV=development
```

- `FLASK_APP`: Menunjukkan nama file utama server.
- `FLASK_ENV`: Menentukan lingkungan pengembangan atau produksi.

2.14 Mengembalikan JSON

Anda dapat mengembalikan objek yang dapat diserialisasi seperti dictionary.

```
    return {"message": "Hello, World!"}
```

JSON adalah format data yang sering digunakan untuk pertukaran data antara server dan klien.

2.15 Opsi Konfigurasi di Flask

Variabel Konfigurasi adalah beberapa opsi konfigurasi yang dapat digunakan:

- `ENV`: Menunjukkan lingkungan (produksi atau pengembangan).
- `DEBUG`: Mengaktifkan mode debug.
- `SECRET_KEY`: Digunakan untuk menandatangani cookie sesi.

2.16 Struktur Aplikasi

- Struktur Direktori Umum

  Berikut adalah contoh struktur direktori untuk aplikasi Flask:

```
my_flask_app/
│
├── app/
│   ├── __init__.py
│   ├── routes.py
│   ├── models.py
│   ├── forms.py
│   ├── static/
│   │   ├── css/
│   │   ├── js/
│   │   └── images/
│   └── templates/
│       ├── layout.html
│       └── index.html
│
├── config.py
├── run.py
└── requirements.txt
```

Figure 1: Struktur Aplikasi Flask

- Penjelasan Setiap Komponen
  - `my_flask_app/`: Ini adalah direktori utama untuk aplikasi Anda.
  - `app/`: Direktori ini berisi semua kode sumber aplikasi Flask Anda.
    - `__init__.py`: File ini digunakan untuk menginisialisasi aplikasi Flask dan mengatur konfigurasi. Di sini, Anda juga dapat mengimpor rute dan model.
    - `routes.py`: File ini berisi semua rute (URL handlers) untuk aplikasi Anda. Di sinilah Anda mendefinisikan fungsi yang akan dijalankan ketika pengguna mengakses URL tertentu.
    - `models.py`: File ini berisi definisi model data, biasanya digunakan dengan ORM (Object-Relational Mapping) seperti SQLAlchemy untuk berinteraksi dengan database.
    - `forms.py`: File ini berisi definisi formulir yang digunakan dalam aplikasi, sering kali menggunakan Flask-WTF untuk menangani formulir dengan lebih mudah.
    - `static/`: Direktori ini menyimpan semua file statis seperti CSS, JavaScript, dan gambar. File-file ini dapat diakses langsung oleh klien.
    - `templates/`: Direktori ini berisi semua template HTML yang digunakan untuk merender halaman web. Flask menggunakan Jinja2 sebagai mesin templating, yang memungkinkan Anda untuk menyisipkan data dinamis ke dalam HTML.
  - `config.py`: File ini berisi konfigurasi aplikasi, seperti pengaturan database, kunci rahasia, dan opsi lainnya. Anda dapat mengatur berbagai konfigurasi untuk lingkungan pengembangan dan produksi di sini.
  - `run.py`: File ini adalah titik masuk untuk menjalankan aplikasi Flask. Anda dapat mengeksekusi file ini untuk memulai server Flask.
  - `requirements.txt`: File ini berisi daftar semua dependensi yang diperlukan untuk aplikasi Anda. Anda dapat menggunakan `pip install -r requirements.txt` untuk menginstal semua paket yang diperlukan.

3 Creating AI Application and Deploy using Flask