Project Setup And Architecture

```
class Intern:
    def _init_(self, name, email, skills):
        self.name = name
        self.email = email
        self.skills = skills

class Internship:
    def _init_(self, title, description, requirements):
        self.title = title
        self.description = description
        self.requirements = requirements

class SmartInternz:
    def _init_(self):
        self.interns = []
        self.internships = []

    def add_intern(self, intern):
        self.interns.append(intern)

    def add_internship(self, internship):
        self.internships.append(internship)

    def match_interns(self):
        for internship in self.internships:
```

```
            print(f"Internship: {internship.title}")
            for intern in self.interns:
                if any(skill in internship.requirements for skill in intern.skills):
                    print(f"  - {intern.name} ({intern.email})")
```

Backend Core Functionalities

```
class User:
    def _init_(self, id, username, email, password):
        self.id = id
        self.username = username
        self.email = email
        self.password = password
class UserManager:
    def _init_(self):
        self.users = []
    def create_user(self, username, email, password):
        new_user = User(len(self.users) + 1, username, email, password)
        self.users.append(new_user)
        return new_user
    def get_user(self, id):
        for user in self.users:
            if user.id == id:
                return user
        return None
    def update_user(self, id, username, email, password):
        user = self.get_user(id)
```

```
        if user:
            user.username = username
            user.email = email
            user.password = password
            return user
        return None
    def delete_user(self, id):
        user = self.get_user(id)
        if user:
            self.users.remove(user)
            return True
        return False
class Authenticator:
    def __
```

Data Handling And Logic

Here's a basic data handling and logic code using Python:

```
import pandas as pd
class DataHandler:
    def _init_(self, data):
        self.data = pd.DataFrame(data)
    def filter_data(self, column, value):
        return self.data[self.data[column] == value]
    def sort_data(self, column):
        return self.data.sort_values(by=column)
    def group_data(self, column):
```

```
        return self.data.groupby(column).size()
data = {
    "Name": ["John", "Jane", "Bob", "Alice"],
    "Age": [25, 30, 35, 20],
    "City": ["New York", "Los Angeles", "Chicago", "New York"]
}

data_handler = DataHandler(data)
filtered_data = data_handler.filter_data("City", "New York")
print(filtered_data)
class BusinessLogic:
    def _init_(self, data_handler):
        self.data_handler = data_handler
    def calculate_average_age(self):
        return self.data_handler.data["Age"].mean()

    def get_oldest_person(self):
        return self.data_handler.data.loc[self.data_handler.data["Age"].idxmax()]
    def get_youngest_person(self):
        return self.data_handler.data.loc[self.data_handler.data["Age"].idxmin()]
business_logic = BusinessLogic(data_handler)
average_age = business_logic.calculate_average_age()
print(f"Average age: {average_age}")
class DataValidator:
    def _init_(self, data):
        self.data = data
```

```
    def validate_age(self):
        for age in self.data["Age"]:
            if age < 0:
                return False
        return True
```

Frontend Development

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
app = dash.Dash(_name_)
app.layout = html.Div([
    html.H1("My App"),
    dcc.Input(id="input", type="text", placeholder="Enter text"),
    html.Div(id="output")
])
@app.callback(
    Output("output", "children"),
    [Input("input", "value")]
)
def update_output(value):
    return f"You entered: {value}"

if _name_ == "_main_":
    app.run_server()
from flask import Flask, render_template, request
```

```
app = Flask(_name_)
@app.route("/")
def index():
    return render_template("index.html")
@app.route("/submit", methods=["POST"])
def submit():
    data = request.form["data"]
    return f"You entered: {data}"
if _name_ == "_main_":
    app.run()
```

Integration And Testing

```
import unittest
from unittest.mock import patch
from my_app import app, db


class TestMyApp(unittest.TestCase):
    def setUp(self):
        app.config["TESTING"] = True
        app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///:memory:"
        db.create_all()
```

Refinement And Deployement

```
import os
def deploy_app():
    os.system("python setup.py build")
 os.system("gunicorn my_app:app --workers 4 --bind 0.0.0.0:5000")
```

```
if _name_ == "_main_":
   deploy_app()
```

Develop The Core Functionalities

```
import requests
from bs4 import BeautifulSoup
def scrape(url):
   response = requests.get(url)
   soup = BeautifulSoup(response.text, 'html.parser')
   return data
import nltk
from nltk.stem import WordNetLemmatizer
def chatbot(message):
   response = "Hello! How can I assist you?"
   return response
import os
def manage_files(directory):
   files = os.listdir(directory)
   # Perform operations
   return files
import pandas as pd
def analyze_data(dataframe):
   summary = dataframe.describe()
   return summary
```

```
def game_loop():
    pygame.init()
    # Game logic
    pygame.quit()
```

Writing The Main Application Logic in App.Py

```
import logging
Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(_name_)
def main():
    try:
        logger.info("Application started")
        logger.info("Application finished successfully")
    except Exception as e:
        logger.error(f"Application failed: {str(e)}")
if _name_ == "_main_":
    main()
```

Deployment

```
from flask import Flask
app = Flask(_name_)
@app.route("/")
def hello_world():
    return "Hello, World!"


if _name_ == "_main_":
```

```
app.run()
```