Date: -10-06-24
Submitted by:- P.likhitha &21kq1a0223

Details of project:I'm implementing this project by using python programming language.

Course Navigator Pro

DESCRIPTION:-

The purpose of "course navigator pro" is used to implement in various types of coding languages like C,Python,Java,c++…..etc.This type of project can develop our skills as well as improving knowledge.it is used to create by our own input and requirements. This project is used in educational institutions and we can also use for our personal profession.educational institutions can use the project as a platform for offering various courses for students and tracking student progress and providing more awareness of various courses.overall,the project can enhance our knowledge,skill devolpment in various languages,this platform can use for individual learns to educational institutions.

REQUIREMENTS:-

User Authentication:-	User should be able to login .
Course:-	Courses which are taken as input from the user.
Course details:-	Details about selected course so that user can prefer suitable course.
Course enrolment:-	Users can enroll in the available courses.
Course duration:-	Duration is based on user selected input.
Expenditure:-	Based on the course.
Course admitted members:-	Based on the user selected course.
Course slots:-	Slots are predefined

#if the adimitted members(course selected by user)more than 250 members alot another slot.

Project Approach:

1. **Requirement Analysis:**
   - Conduct stakeholder interviews and surveys to gather requirements from learners, instructors, educational institutions, and industry stakeholders.
   - Identify key features, user personas, use cases, and technical specifications for the Course Navigator Pro platform.
2. **Design and Prototyping:**
   - Create wireframes, mockups, and interactive prototypes to visualize the user interface, navigation flow, and feature interactions.
   - Design the database schema, API architecture, and system components for scalability, performance, and security.
3. **Development Iterations:**
   - Adopt an Agile development methodology with iterative sprints to prioritize and implement features incrementally.
   - Develop modular, reusable components using modern web technologies such as React.js for the front end and Node.js for the back end.
4. **Testing and Quality Assurance:**
   - Conduct unit tests, integration tests, and end-to-end tests to ensure the functionality, usability, and reliability of the platform.
   - Perform usability testing with real users to gather feedback and identify usability issues or pain points for improvement.

5. **Deployment and Launch:**
    - Deploy the Course Navigator Pro platform to a scalable and secure cloud infrastructure such as AWS, Azure, or Google Cloud Platform.
    - Configure monitoring, logging, and error tracking systems to monitor system health and performance in real-time.
6. **User Training and Support:**
    - Provide comprehensive documentation, tutorials, and onboarding resources to guide users in navigating and utilizing the platform effectively.
    - Offer responsive customer support channels such as email, chat support, and community forums to address user inquiries and technical issues.
7. **Continuous Improvement:**
    - Gather feedback from users through surveys, user analytics, and user interviews to identify areas for improvement and feature enhancements.
    - Regularly release updates, bug fixes, and new features based on user feedback, industry trends, and emerging technologies.

project code:

```
courses = []
l=[]
def add_course(c_name, c_duration, c_price, a_members):
    course = {
        'c_name': c_name,
        'c_duration': c_duration,
        'c_price': c_price,
        'a_members': a_members
    }
    courses.append(course)
def print_separator():
    print("+------------------+------------------+---------------+------------------+")
def check_course_exist(c_name):
    for course in courses:
        if course['c_name'] == c_name:
            return True
    return False
def courses_with_less_than_50_seats():
    count = 0
    for course in courses:
        if course['a_members'] < 50:
            count += 1
    return count
def courses_with_price_above_1000():
    count = 0
    for course in courses:
        if course['c_price'] > 1000:
            count += 1
    return count
def total_courses():
    return len(courses)
def total_seats():
    total = 0
    for course in courses:
        total += course['a_members']
    return total
n = int(input("Enter the number of courses: "))
```

```
for _ in range(n):
    c_name = input("Enter course name: ")
    c_duration = int(input("Enter course duration (in weeks): "))
    c_price = int(input("Enter course price: "))
    a_members = int(input("Enter admitted members: "))
    add_course(c_name, c_duration, c_price, a_members)
print_separator()
print("|   Course Name    | Duration (weeks) | Price ($) | Admitted Members |")
print_separator()

for course in courses:
    print("| {:<16} | {:<16} | {:<10} | {:<16} |".format(
        course['c_name'],
        course['c_duration'],
        course['c_price'],
        course['a_members']
    ))
print_separator()
print("Second course name:", courses[1]['c_name'])
course_name_to_check = input("Enter the course name to check if it exists: ")
if check_course_exist(course_name_to_check):
    print(f"The course '{course_name_to_check}' exists.")
else:
    print(f"The course '{course_name_to_check}' does not exist.")
print("Courses with less than 50 seats:", courses_with_less_than_50_seats())
print("Courses with price above 1000:", courses_with_price_above_1000())
print("Total number of courses:", total_courses())
print("Total number of seats in the college:", total_seats())
sorted_courses_by_name = sorted(courses, key=lambda x: x['c_name'])
print("Courses sorted by name:")
for course in sorted_courses_by_name:
    print(course['c_name'])
sorted_prices = sorted([course['c_price'] for course in courses])
print("\nPrices sorted:")
for price in sorted_prices:
    print(price)
```

Sample output:-

```
Enter the number of courses: 20
Enter course name: c
Enter course duration (in weeks): 3
Enter course price: 7000
Enter admitted members: 34
Enter course name: python
Enter course duration (in weeks): 7
Enter course price: 8000
Enter admitted members: 56
Enter course name: java
Enter course duration (in weeks): 9
Enter course price: 8000
Enter admitted members: 67
Enter course name: c++
Enter course duration (in weeks): 6
Enter course price: 6000
Enter admitted members: 25
Enter course name: html
Enter course duration (in weeks): 6
Enter course price: 8000
Enter admitted members: 56
Enter course name: java full stack
Enter course duration (in weeks): 6
Enter course price: 10000
Enter admitted members: 45
Enter course name: sql
Enter course duration (in weeks): 6
Enter course price: 6000
Enter admitted members: 56
Enter course name: dbms
Enter course duration (in weeks): 6
Enter course price: 5000
Enter admitted members: 67
```

```
Enter admitted members: 67
Enter course name: oops
Enter course duration (in weeks): 9
Enter course price: 12000
Enter admitted members: 45
Enter course name: full stack
Enter course duration (in weeks): 5
Enter course price: 6000
Enter admitted members: 34
Enter course name: python full stack
Enter course duration (in weeks): 7
Enter course price: 8000
Enter admitted members: 65
Enter course name: os
Enter course duration (in weeks): 8
Enter course price: 6500
Enter admitted members: 54
Enter course name: perl
Enter course duration (in weeks): 5
Enter course price: 8000
Enter admitted members: 54
Enter course name: visual basic
Enter course duration (in weeks): 5
Enter course price: 6000
Enter admitted members: 43
Enter course name: copol
Enter course duration (in weeks): 4
Enter course price: 5000
Enter admitted members: 34
Enter course name: nosql
Enter course duration (in weeks): 4
Enter course price: 7000
Enter admitted members: 23
Enter course name: c-sharp
Enter course duration (in weeks): 6
Enter course price: 6000
Enter admitted members: 46
```

```
Enter course name: rust
Enter course duration (in weeks): 5
Enter course price: 5000
Enter admitted members: 45
Enter course name: c#
Enter course duration (in weeks): 7
Enter course price: 8000
Enter admitted members: 56
Enter course name: kotlin
Enter course duration (in weeks): 4
Enter course price: 6000
Enter admitted members: 67
```

Course Name	Duration (weeks)	Price ($)	Admitted Members
c	3	7000	34
python	7	8000	56
java	9	8000	67
c++	6	6000	25
html	6	8000	56
java full stack	6	10000	45
sql	6	6000	56
dbms	6	5000	67
oops	9	12000	45
full stack	5	6000	34
python full stack	7	8000	65
os	8	6500	54
perl	5	8000	54
visual basic	5	6000	43
copol	4	5000	34
nosql	4	7000	23
c-sharp	6	6000	46
rust	5	5000	45
c#	7	8000	56
kotlin	4	6000	67

```
Second course name: python
Enter the course name to check if it exists: nosql
The course 'nosql' exists.
Courses with less than 50 seats: 10
Courses with price above 1000: 20
Total number of courses: 20
Total number of seats in the college: 972
Courses sorted by name:
c
c#
c++
c-sharp
copol
dbms
full stack
html
java
java full stack
kotlin
nosql
oops
os
perl
python
python full stack
rust
sql
visual basic

Prices sorted:
5000
5000
5000
6000
6000
6000
6000
```

```
Prices sorted:
5000
5000
5000
6000
6000
6000
6000
6000
6000
6500
7000
7000
8000
8000
8000
8000
8000
8000
10000
12000
```

Explanation:-

1. **add_course**: This function takes input for a new course, including its name, duration, price, and number of admitted members, and adds it to the `courses` list.
2. **print_separator**: This function prints a separator line for formatting purposes in the output.
3. **check_course_exist**: This function checks if a course with a given name already exists in the `courses` list. It returns `True` if the course exists, otherwise `False`.
4. **seats_left**: This function returns the number of seats left for a given course name. It searches for the course in the `courses` list and returns the value of 'a_members'.

5. **courses_with_less_than_50_seats**: This function counts the number of courses with less than 50 seats available. It iterates through the `courses` list and increments the count if the number of admitted members is less than 50.
6. **courses_with_price_above_1000**: This function counts the number of courses with a price above $1000. It iterates through the `courses` list and increments the count if the price is above 1000.
7. **total_courses**: This function returns the total number of courses in the `courses` list.
8. **total_seats**: This function calculates and returns the total number of seats in all courses combined.

After defining these functions, the script prompts the user to input details for multiple courses, then prints the courses' information in a formatted table. It also provides functionality to check if a course exists, check the number of seats left for a course, and gives various statistics about the courses, such as the number of courses with less than 50 seats or the number of courses with a price above $1000.

Conclusion:

By following this project approach, Course Navigator Pro can become a leading platform for coding language education, empowering learners worldwide to acquire valuable skills, advance their careers, and contribute to the thriving community of developers and technologists.