Final Year Design Project

AI-Powered Course Recommendation System

By

Student Name 1 BXXF21Mxxx

Student Name 2 BXXF21Mxxx

Student Name 3 BXXF21Mxxx

Student Name 4 BXXF21Mxxx

Under the supervision of

Prof. Dr. Engr. Shahzad Sarwar

Bachelor of Science in [department name] (2021-2025)

FACULTY OF COMPUTING &

INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE.

AI-Powered Course Recommendation System

A project presented to

University of the Punjab, Lahore

In partial fulfilment

of the requirement for the degree of

Bachelors of Science in [Department] (2021-2025)

By

Student Name 1 BXXF21Mxxx

Student Name 2 BXXF21Mxxx

Student Name 3 BXXF21Mxxx

Student Name 4 BXXF21Mxxx

FACULTY OF COMPUTING &

INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE

DECLARATION

We hereby declare that this software, neither whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other, we will stand by the consequences. No portion of the work presented has been submitted of any application for any other degree or qualification of this or any other university or institute of learning.

Signature: ----------

Student Name 1[Roll No]

Signature: ----------

Student Name 3 [Roll No]

Signature: ----------

Student Name 2 [Roll No]

Signature: ----------

Student Name 4 [Roll No]

CERTIFICATE OF APPROVAL

It is to certify that the final year design project (FYDP) of BSXX "Project title" was developed by **STUDENT NAME 1 (BXXF21Mxxx), STUDENT NAME 2 (BXXF21Mxxx), STUDENT NAME 3 (BXXF21Mxxx),** and **STUDENT NAME 4 (BXXF21Mxxx)** under the supervision of "SUPERVISOR NAME" in my opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in [Department name.]

Signature: ----------

FYDP Supervisor:

Signatures (Faculty Advisory Committee (FAC)

Signatures			
Name			
	FAC1	FAC2	FAC3

Signature: ----------

Head of FYDP Coordination Office:

Signature:---------- Dated: _____

Chairperson, Department of [Department name]

Executive Summary

Executive summary serves to provide a quick understanding of the project's objectives, methodology, and conclusions. It must be concise, clearly written, and proofread.
[*Objectives entails questions being addressed or the problems we are trying to solve. Methodology describes the summary of the approach being used to solve the problem. Conclusion specifies findings from the project and its implications. (It should not be more than one page long. Ideally written after completing the project and is a true reflection of the project done)]*

Keywords: Specify atleast three keywords

Acknowledgement

The acknowledgment section is an opportunity to express gratitude and appreciation to individuals and organizations who have contributed to the completion of your Final Year Project (FYP).

Signature: ----------

Student Name 1 [Roll No]

Signature: ----------

Student Name 3[Roll No]

Signature: ----------

Student Name 2 [Roll No]

Signature: ----------

Student Name 4[Roll No]

Abbreviations

Table of all the abbreviations and acronyms used in your FYP along with their respective brief description. The abbreviation list must be ordered alphabetically.

BL	Backlog list : List of the requirements under consideration
FCIT	Faculty of Computing and Information Technology

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1. Introduction

This chapter provides the overview of the project. The first paragraph should provide the chapter introduction – presenting the purpose of the chapter and how this chapter contributes to achieve project goals.

1.1 Problem Statement

Provide a problem statement in a concise paragraph describing
Why are you developing this software system?
What problem your software system is going to solve?
[Usually in 10-16 sentences]

1.2 Problem Solution

Describe the application of software being specified including objectives and goals keeping in view the problem statement listed in Section 2.1
[Provide list of objectives]

[Usually in 14-16 sentences]

1.3 Objectives of the Proposed System

Quantifiable objectives to be achieved using this system. Please follow the do's and don'ts of writing objectives

• **Be Specific**:
  • Clearly define what the software is intended to achieve.
  • Use precise language to avoid ambiguity.
• **Be Measurable**:
  • Include metrics or criteria that can be used to assess whether the objectives have been met.
  • Example: "Reduce user login time by 50%."
• **Be Achievable**:
	• **Don't Be Vague**:
	  • Avoid unclear or ambiguous statements that can be interpreted in multiple ways.
	  • Example of vague objective: "Improve system performance."
	• **Don't Overload with Jargon**:
	  • Use clear and simple language that all stakeholders can understand.
	  • Avoid technical jargon that may confuse non-technical stakeholders.
	• **Don't Set Unreachable Goals**:

- Set realistic goals that can be accomplished within the constraints of time, budget, and resources.
- Avoid overly ambitious objectives that are unlikely to be met.

- **Be Relevant**:
  - Ensure that the objectives align with the overall goals and priorities of the organization or project.
  - Each objective should contribute to the success of the software.
- **Be Time-bound**:
  - Specify a timeline or deadline for achieving the objectives.
  - Example: "Complete the user interface redesign by Q3 2024."
- **Be User-focused**:
  - Consider the needs and expectations of the end users.
  - Objectives should aim to improve user experience and satisfaction.
- **Prioritize Objectives**:
  - Rank objectives based on their importance and impact.
  - Focus on the most critical objectives first.
- **Get Stakeholder Input**:
  - Involve key stakeholders in defining the objectives to ensure they meet the needs and expectations of all parties involved.
  - Regularly review and update objectives as needed based on stakeholder feedback

- Ensure objectives are realistic and can be achieved with the available resources and constraints.
- Overly ambitious goals can lead to frustration and project failure.

- **Don't Ignore Constraints**:
  - Take into account any limitations or constraints that might affect the achievement of the objectives.
  - This includes budget, time, technology, and manpower.
- **Don't Forget to Review and Revise**:
  - Objectives should be reviewed periodically and revised as needed based on project progress and any changes in scope or requirements.
  - Static objectives may become irrelevant or unachievable over time.
- **Don't Neglect Documentation**:
  - Document the objectives clearly and share them with all relevant stakeholders.
  - Proper documentation ensures everyone is on the same page.
- **Don't Ignore Risks**:
  - Consider potential risks and challenges that could impact the achievement of objectives.
  - Plan for contingencies to address these risks.
- **Don't Overlook Dependencies**:
  - Recognize any dependencies between objectives and address them in your planning.

	• Ensure that achieving one objective does not negatively impact another.

1.4 Scope

Scope defines the boundaries and range of the proposed solution, that what would be the part of your project. Write down in logical flow with consistency. Make sure that the scope content must be:

1. Clear and Specific
2. Deliverables are defined
3. System boundaries are clearly presented. Clearly specifies delimitations to avoid software creep
4. Identify stakeholders
5. Must align with objectives
6. Set priorities to clearly prioritize key features and functionalities.
7. Consider constraints like time, technology and resources
8. Provide milestones

1.5 System Components

Write down the modules/system components of the proposed project.
Each module should highlight features, using bulleted/numbered notation.
When developing both a mobile app and a web app, group the modules according to the system types, such as, Client Web App, Client Mobile App, Admin Web App etc.

1.5.1 Module 1: Module Name

Write down the List of features associated with this module/component.

1.6 Related System Analysis/Literature Review

Briefly provide an analysis of the related system which may help you to specify the contribution of the proposed project. This includes the characteristics of the existing/similar systems related to your proposed project. Don't use more than 4 sentences for explaining a single system/application. Highlight its characteristics including the features lacking in the existing systems that may be present in your system.

Table 1- 1 Related System Analysis with proposed project solution

Application Name	**Weakness**	**Proposed Project Solution**
The name of related application(s).	Weaknesses may include limited features, low quality functionality and processes.	The way the proposed project mitigates the weaknesses.

1.7 Vision Statement

Write a concise vision statement that summarises the long-term purpose, intent and significance of the product. The vision statement reflects a balanced view that will satisfy the expectations of diverse stakeholders. It can be somewhat idealistic but should be grounded in the realities of existing or anticipated markets. The following keyword template works well for crafting a product vision statement:

For [target customer]

Who [statement of the need or opportunity]

The [product name]

Is [product category]

That [major capabilities, key benefit, compelling reason to buy or use]

Unlike [primary competitive alternative, current system, current business process]

Our product [statement of primary differentiation and advantages of new product]

1.8 System Limitations and Constraints

Write down the limitations and constraints of the proposed project. Limitations refers to the external factors which are beyond our control (like technological restrictions), whereas constraints refer to the self-imposed controllable factors. Examples of constraints include scope of the project.

1.9 Tools and Technologies

Mention all the hardware/software tools and technologies with version information used in implementation of the project. Write about the APIs, language(s), SDK(s) etc. which you will use for implementation.

Example:

Table 1- 2Tools and Technologies for Proposed Project

Tools And Technologies	**Tools**	**Version**	**Rationale**
	Technology	**Version**	**Rationale**

1.10 Project Deliverables

List down the project deliverables. This include planning document, SRS, Design, Prototypes, project, project report. (these documents will be added as appendices of FYP report)

1.11 Project Planning

Gantt Chart representing the plans of carrying out the project activities, representing the milestones as well as resources required for each task

1.12 Summary

Summary of the key points discussed in the chapter. It may also reiterate chapter's importance in addressing the projects objectives

Chapter 2

Requirements Analysis

2. Analysis

Provide an introduction to the chapter in a few lines. Write details in each of the sections provided ahead.

2.1 User classes and characteristics

Identify the various user classes that you anticipate will use this product and describe their pertinent characteristics using 2 column table.

2.2 Requirement Identifying Technique

This section describes the requirements identifying technique(s) which further help to derive functional requirements specification. The selection of the technique(s) will depend on the type of project. For instance,

- **Use case** includes **detailed use case descriptions + use case diagram)** is an effective technique for interactive end-user applications. Use case name and associated requirements must be presented here. However, use case descriptions must be given in fully dressed format.
- **Storyboarding** for graphically intensive applications. Visual representation of sequence of events, designs to represent the flow.

2.3 Functional Requirements

This section describes the functional requirements of the system expressed in the natural language style. This section is typically organized by feature as a system feature name and specific functional requirements associated with this feature. It is just one possible way to arrange them. Other organizational options include arranging functional requirements by use case, process flow, mode of operation, user class, stimulus, and response depend on what kind of technique has been used to understand functional requirements. Hierarchical combinations of these elements are also possible, such as use cases within user classes.

2.3.1 Functional Requirement X

Itemize the specific functional requirements associated with each feature. These are the software capabilities that must be implemented for the user to carry out the feature's services or to perform a use case. Describe how the product should respond to anticipated error conditions and to invalid inputs and actions. Uniquely label each functional requirement, as described earlier. You can create multiple attributes for each functional requirement, such as rationale, source, dependencies, etc. The following template is required to write functional requirements.

Table 1- 1Description of FR-1

Identifier	FR-1
Title	Title of requirement
Requirement	Description of requirement which may be written either from the user or system perspective e.g. If written in a **user perspective** The [user class or actor name] shall be able to [do something] [to some object] [qualifying conditions, response time, or quality statement]. If written in a **system perspective** [optional precondition] [optional trigger event] the system shall [expected system response]
Source	Where this requirement comes from (who originate it)
Rationale	The motivation behind the requirement
Business Rule (if required)	Any restriction, policy, the rule that the particular requirement must be fulfilled through its functional behaviour
Dependencies	Requirements ID that is dependent on this requirement
Priority	High/Medium/Low

2.4 Non-Functional Requirements

This section specifies nonfunctional requirements other than constraints, supporting requirements recorded in section 2.3, and external interface requirements. These quality requirements should be specific, quantitative, and verifiable. The following are some examples of documenting guidelines.

2.4.1 Reliability

Requirements about how often the software fails. The measurement is often expressed in MTBF (mean time between failures). Be sure to specify the consequences of software failure, how to protect from failure, a strategy for error detection, and a strategy for correction.

2.4.1 Usability

Usability requirements deal with ease of learning, ease of use, error avoidance and recovery, the efficiency of interactions, and accessibility. The usability requirements specified here will help the user interface designer create the optimum user experience. It may be quantified using the concepts of effort required to perform a task in terms of number of interactions required, considerations and solutions to make an accessible solution.

2.4.2 Performance

State specific performance requirements for various system operations. It may include response time or expected resources utilization requirements. If different functional requirements or features have different performance requirements, it's appropriate to specify those performance goals right with the corresponding functional requirements, rather than collecting them in this section.

2.4.3 Security

One or more requirements about protection of your system and its data. The measurement can be expressed in a variety of ways (effort, skill level, and time) to break into the system. Do not discuss solutions (e.g. passwords) in a requirements document.

2.5 External Interface Requirements

This section provides information to ensure that the system will communicate properly with users and with external hardware or software elements. A complex system with multiple subcomponents should create a separate interface specification or system architecture specification. The interface documentation could incorporate material from other documents by reference. For instance, it could point to a hardware device manual that lists the error codes that the device could send to the software.

2.5.1 User Interfaces Requirements

Describe the logical characteristics of each user interface that the system needs. Some possible items to include are

- References to GUI standards or product family style guides that are to be followed.
- Standards for fonts, icons, button labels, images, colour schemes, field tabbing sequences, commonly used controls, and the like.
- Screen layout or resolution constraints.
- Standard buttons, functions, or navigation links that will appear on every screen, such as a help button.
- Shortcut keys.
- Message display conventions.
- Layout standards to facilitate software localization.
- Accommodations for visually impaired users.

Document the user interface design details, such as specific dialog box layouts, in a separate user interface specification, not in the SRS. Including screen mock-ups in the SRS to communicate another view of the requirements is helpful but make it clear that the mock-ups are not the committed screen designs. If the SRS is specifying an enhancement to an existing system, it sometimes makes sense to include screen displays exactly as they are to be implemented. The developers are already constrained by the current reality of the existing system, so it's possible to know up front just what the modified, and perhaps the new, screens should look like.

2.5.2 Software interfaces

Describe the connections between this product and other software components (identified by name and version), including other applications, databases, operating systems, tools, libraries, websites, and integrated commercial components (If any).

2.5.3 Hardware interfaces

Describe the characteristics of each interface between the software components and hardware components, if any, of the system. This description might include the supported device types, the data and control interactions between the software and the hardware, and the communication protocols to be used.

2.5.4 Communications interfaces

State the requirements for any communication functions the product will use, including email, web browser, network protocols, and electronic forms.

2.6 Summary

Summary of the processes involved in requirements identification and uses case analysis. It may also reiterate requirements engineering process's importance in addressing the projects objectives.

Chapter 3

Design and Architecture

3. System Design

Explain the product perspective in which software will operate including dependencies and interaction with other system. List the design constraints (like performance requirements) impacting design.

3.1 Design considerations

Assumptions and Dependencies: Lists any assumptions and dependencies that influence the design decisions.

Limitations: Describes any limitations in the software that could affect its functionality, performance, or compatibility.

Risks: Identifies potential risks and how they were addressed in the design

3.2 Design Models

Provide the descriptions of the following models used to describe the system design. Also ensure visibility of all diagrams.

The applicable models for the project using object-oriented development approach may include:

1. Class Diagram
2. Interaction Diagram (Either sequence or collaboration)
3. State Transition Diagram (for the projects which include event handling and backend processes)

3.3 Architectural Design

Develop a component structure and explain the relationships between these components to achieve the complete functionality of the system. This is a high-level overview of system's inter-modules collaboration to achieve the desired functionality.

Provide a diagram showing the major subsystems and their connections.

- UML Class relationship diagram and UML Component diagram
- After finalizing architecture style/pattern diagram (MVC, Client-Server, Layered, Multi-tiered) create a detailed mapping modules/components to each part of the architecture

3.4 Data Design

Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed, and organized. This includes the dataset/ database design and/or other design of other data structures used.

3.4.1 Data Dictionary

- **Alphabetical List of System Entities or Major Data with Types and Descriptions:**
    - Prepare an alphabetical list of all system entities or significant data. For each entry, include the **data type** and provide a brief **description** of its purpose within the system.
    - Present this information in a **two-column format**:
        - The **first column** will contain the **terminology** (e.g., object name, attribute, method, or method parameter).
        - The **second column** will provide the **description**, including data types, roles, or any other relevant information

- **Structured Approach: Functions and Function Parameters:**
    - For systems following a structured programming paradigm, list **all functions** along with their **function parameters**. For each function, provide a description detailing its functionality, and for each parameter, specify the **data type** and its role within the function.

- **Object-Oriented (OO) Approach: Objects, Attributes, Methods, & Method Parameters:**
    - For systems using the Object-Oriented (OO) approach, list the **objects** and their **attributes**, followed by a description. Additionally, list the **methods** associated with each object, along with their **method parameters**.

3.5 User Interface Design

Describe overview of the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user. [See the User Interface Design in Appendix -]

3.3.1 Screen Images

Display few screenshots showing the interface from the user's perspective. These can be hand-drawn, or you can use Prototyping tool like Canva/ Figma. Just make them as accurate as possible.

3.3.2 Screen Objects and Actions

A discussion of screen objects and actions associated with those object for two use cases.

3.4 Behavioural Model

Interaction Diagrams: Includes diagrams (Sequence or Collaboration diagram) that illustrate interactions among components. Make sure that objects & methods mentioned in the interaction diagram are consistent with the class diagram

State Diagrams: Shows state transitions within the software, especially useful for systems with complex workflows or lifecycle states.

Make sure that each diagram has assigned a proper caption, so that table of figures can be created.

3.5 Design Decisions

Highlight the design choices while designing your system (e.g. choosing the Object oriented design pattern, choosing between the algorithmic approaches, normalization level of the database etc). Clearly specify which approach you have used and why you have made a specific design choice.

3.6 Summary

Summary of the key design ideas discussed, design refinement and decisions taken in the chapter. It may also reiterate chapter's importance in addressing the projects objectives

Chapter 4

Implementation

4. Implementation

This chapter discusses implementation details of the project. **Do not** put your source code here, however, you are required to write the core components functionalities in pseudocode form (Following sections are required in this chapter).

Note: You are required to follow proper coding standard to write your source code. For guidelines, **General Coding Standards & Guidelines** are provided in Appendix D.

4.1 Algorithm

Mention the algorithm(s) and its complexity used in your project to get the work done with regards to major modules. Provide a pseudocode explanation regarding the functioning of the core features. Be sure to use the correct syntax and semantics for algorithm representations. Following are few examples of algorithms/pseudocode:

Table 5- 1 Example of Algorithm

Algorithm 1 MHCF component
Input:
Output:
1: merge ← true 2: Flag ← false 3: **While**(merge==’true’) **do:** 4: merge ← false 5: **for** i in range (0: len(G)-1): 6: **for** j in range (i+1: len(G)):

4.2 External APIs/SDKs

Describe the third-party APIs/SDKs used in the project implementation in the following table. Few examples of APIs are provided in the table.

Table 5- 1 Details of APIs used in the project

Name of API and version	**Description of API**	**Purpose of usage**	**List down the API endpoint/function/class in which it is used**

4.3 Code Repository

In order to manage the version control and collaboration effectively for the group project, Git will be used as the primary tool. All project files, including code, documentation, and other relevant resources, will be stored in the Git repository. This will ensure proper tracking of changes, collaboration among team members, and access to the most up-to-date version of the project.

Git Repository Link:
The repository for the group project can be accessed using the following link:
[Git repository link]

4.3.1 Metrics of the Git Repository:

The following metrics will be monitored to ensure effective use of the Git repository:

Commits: The number of commits made to track the frequency and consistency of contributions by team members.

Branches: The number of active and merged branches, representing the organization of different features and stages of development.

Pull Requests: The number of pull requests created, merged, or in review, showing the collaboration and code review process.

Issues: The number of open and closed issues, indicating bug tracking, feature requests, or task management.

Contributors: A list of contributors and their contribution statistics (commits, lines added, and lines removed).

Code Reviews: The number of reviews conducted on pull requests to ensure code quality and compliance with project standards.

4.4 Summary

Summary of the key points discussed in the chapter. It may also reiterate chapter's importance in addressing the projects objectives

Chapter 5

Testing and Evaluation

5. Introduction

In order to ensure software verification and validation, software testing is conducted. Though it is preferred to specify automated test scripts, testcases referring to manual testing can also be presented. Test scripts must conform to the following best practices:

Keep Tests Independent: Ensure each test can run independently of others.
Use Meaningful Assertions: Make sure your assertions are clear and meaningful.
Handle Test Data: Use setup and teardown methods to manage test data.
Avoid Hardcoding: Use variables and configurations to avoid hardcoding values.
Make Tests Repeatable: Ensure that tests can be run multiple times without failure due to state persistence.

Follow the template (represented in Table-1) for documenting the testcases of each of these categories

5.1 Unit Testing (UT)

It's a level of software testing where individual units of a software/component are tested. The purpose is to validate that each unit of the software performs as designed.

5.2 Functional Testing (FT)

In this functional testing, the functionality of each of the module/ stakeholders' view is tested. This is to ensure that the system produced meets the specifications and requirements

5.3 Integration Testing (IT)

After ensuring the individual test runs successfully, the interaction of the classes/ functions is tested to assess whether the feature is yielding the desired results.

5.4 Performance Testing (PT)

In order to assess the performance of the feature, the performance test is conducted. It usually measures response time, stability (reliability under stress). Specify the tool used for conducting the performance testing. Specify the screen shots (2-4) representing the configuration as well as output of the performance testing conducted.

Table 1- Testcase

Testcase ID	Unique identifier of testcase (e.g. UT1 for Unit testcase1)
Requirement ID	Requirement ID of the requirement to be tested
Title	Feature name or requirement to be tested must be represented in brief
Description	Description of feature to be tested
Objective	Why this testcase is needed
Driver/precondition	Setup required before test execution

Test steps	Step by step instructions to test
Input	Input to be provided to the testcase
Expected Results	Expected outcome of the testcase
Actual Result	Actual outcome of the testcase
Remarks	Test Status (Fail/Pass)

5.5 Summary

Summary of the key points discussed in the chapter. It may also reiterate chapter's importance in addressing the projects objectives

Chapter 6

System Conversion

6. Introduction

System conversion is the process of changing from one information system to another. This involves transferring data, processes, and operations from an old system to a new one.

6.1 Conversion Method

Choose and specify the conversion method for your project. There are several methods for converting systems:

Direct Conversion:
This method involves abruptly stopping the old system and immediately starting the new one. It's risky but can be cost-effective.
Parallel Conversion:
Both the old and new systems operate simultaneously for a period, allowing for comparison and verification. This is less risky but more expensive
Pilot Conversion:
The new system is implemented in a limited part of the organization, allowing for testing and adjustments before full implementation.
Phased Conversion:
The new system is implemented in stages, allowing for gradual adjustments

6.2 Deployment

This may include building the application, deployment, restoring a database from its backup, configuration to put the application in operational usage. It also includes the steps to start the application and checking successful deployment.

It is expected to present the steps required to make the software operational on the customer site. If the project is not for real clients, students are expected to mention the steps required to deploy it on docker.

6.2.1 Data Conversion

Specify the steps used to convert the data from your system to the deployed system. These steps may include data extraction and cleaning from existing system, validating data and loading it into target system. Data backup and restore procedure is one of the examples of the system.

6.2.2 Training

Section specifies user manual to train the users in using the application. User manual of two main use cases of the system can be presented.

6.3 Post Deployment Testing

Steps required to assess that software is properly deployed and converted. This includes running the testcases to ensure correct conversion of system

6.4 Challenges

List down the challenges encountered & their solutions while deploying the system and putting it in the operational usage.

6.5 Summary

Present the summary of the deployment process. Mention the challenges/ issues encountered, also present the ways those issues were resolved.

Chapter 7

Conclusion

7. Introduction

This chapter concludes the project, evaluate the project objectives & goals and highlights future work.

7.1 Evaluation

Make a list of the objectives specified in Chapter 1 and trace whether they have been implemented in your developed FYDP.

Objectives	**Status**
Recommend course in at most 2minutes	Completed

7.2 Traceability Matrix

Shows how each design, code and test configuration item traces back to a specific requirement in the Software Requirements Specification (SRS), ensuring that all requirements are accounted for in the final product. Use table to document requirements traceability, where requirement ID is a unique identifier assigned to the requirement, requirement description refers to brief description of the requirement discussed. Design specification entails list of the components/classes/algorithm where the requirement is addressed. Code represents the name of the file where this functionality is developed. Test ID represent the ID of the testcase documenting test data and procedure.
Make sure that the list is complete and consistent with requirements mentioned in SRS

Requirement ID	**Requirement Description**	**Design Specification**	**Code**	**Test ID**
R1	Recommend Course to the user	Component “Recommend”	Course.aspx	UTI

7.3 Conclusion

Conclude the section with concise summary statement, reinforcing the significance of your project's contributions, its significance and the fulfilment of its objectives agreed in the beginning of the project. Provide the table specifying correlation of objectives defined and functionality provided by the system.

7.4 Future Work

Provide suggestions or recommendations for future work or enhancements to your project. This could include additional features, optimisations, scalability improvements, or other potential applications which were excluded from this project's scope.

References

List any documents or other resources to which this SRS refers, if any. These might include user interface style guides, standards, system requirements specifications, interface specifications, or the SRS for a related product. The following are a few examples of different resources.

Book
Author(s). Book title. Location: Publishing company, year, pp.
Example:
W.K. Chen. Linear Networks and Systems. Belmont, CA: Wadsworth, 1993, pp. 123-35.

Article in a Journal
Author(s). “Article title”. Journal title, vol., pp, date.
Example:
G. Pevere. “Infrared Nation.” The International Journal of Infrared Design, vol. 33, pp. 56-99, Jan. 1979.

Articles from Conference Proceedings (published)
Author(s). “Article title.” Conference proceedings, year, pp.
Example:
D.B. Payne and H.G. Gunhold. “Digital sundials and broadband technology,” in Proc. IOOC-ECOC, 1986, pp. 557-998.

World Wide Web
Author(s)*. “Title.” Internet: complete URL, date updated* [date accessed].
M. Duncan. “Engineering Concepts on Ice. Internet: www.iceengg.edu/staff.html, Oct. 25, 2000 [Nov. 29, 2003].

Appendix A

Use case Description Template

Appendix-A Use Case Description(Fully Dressed Format)

The Table A-1 below indicates a comprehensive use case template

Table A- 1Show the detail use case template and example

Use Case ID:	Enter a unique numeric identifier for the Use Case. e.g. UC-1
Use Case Name:	Enter a short name for the Use Case using an active verb phrase.
Actors:	[An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks.]
Description:	[Provide a brief description of the reason for and outcome of this use case.]
Trigger:	[Identify the event that initiates the use case.]
Preconditions:	[List any activities that must take place, or any conditions that must be true, before the use case can be started.]
Postconditions:	[Describe the state of the system at the conclusion of the use case execution.]
Normal Flow:	[Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions.
Alternative Flows:	[Document legitimate branches from the main flow to handle special conditions (also known as extensions). For each alternative flow reference the branching step number of the normal flow and the condition which must be true for this extension to be executed]
Business Rules	Use cases and business rules are intertwined. Some business rules constrain which roles can perform all or parts of a use case. Perhaps only users who have certain privilege levels can perform specific alternative flows. That is, the rule might impose preconditions that the system must test before letting the user proceed. Business rules can influence specific steps in the normal flow by defining valid input values or dictating how computations are to be performed]
Assumptions:	[List any assumptions].

Appendix-B

Coding Standards

Appendix-B General Coding Standards & Guidelines

1. Follow a consistent variable naming convention throughout the code. E.g.
    - Snakecase (words are delimited by "_" like: variable_one)
    - Pascalcase (words are delimited by capital letters like: VariableOne)
    - Camelcase (words are delimited by capital letters except the initial word like: variableOne)
    - Hungarian Notation (describes the variable type or purpose at the start of the variable name like: arrDistributeGroup)
2. Use naming that visually describes scope like privateField, Const etc
3. Use read only/immutable when a field's value should not be changed after initialization
4. Use only get, for properties that should not be updated from outside
5. Name functions according to their functionalities.
6. Insert appropriate comments to make the code understandable to any reader. Additionally follow a consistent style to do so. E.g.

```
/* the below function will be used for the addition of two variables*/
int Add(){
//logic of the function
}
```

   Avoid commenting on obvious things
7. Make use of indentation for indicating the start and end of the control structures along with a clear specification of where the code is between them.
8. Follow consistent naming convention for files and folders.
9. Follow proper structure for classes
10. Group code entities logically into projects/packages/modules/folders
    a. Separate logical layers of application into different modules/services/utilities etc.
    b. User separate files for each class, struct, interface, enum etc. Name of the file and the enclosing entity must be same. E.g., class Employee in Employee.cs/Employee.java
11. Define and use everything within the minimum scope possible
12. Use proper access modifier for all code entities if required
13. Code entities should have maximum cohesion and least coupling possible.
14. Follow DRY law.
    a. Do not repeat code.
    b. A piece of knowledge should exist only in one place within the codebase/application
    c. Reuse code as much as possible
    d. Always write short methods
    e. Single method should not have too many logic, long conditional flow or too many parameters
15. Strictly follow Single Responsibility Principle (SRP) when writing methods, classes, modules, projects, packages, or any other code entities.
16. Write classes and other code entities that are easy to extend without modification.
17. Handle exceptions
18. Log exception and other significant event details
19. Follow a consistent convention for logging all over the application

Appendix C

Prototype

Appendix-C Application Prototype