Skill Development Tracker
(Xunemp)

A Data Structure Based Project Report

Team Members:

Bhuvan (106124026)
Varun (106124106)
Abhay (106124002)

Under the Guidance of

Mr. Oswald
Faculty, Department of Computer Science and Engineering

National Institute of Technology, Tiruchirappalli

October 30, 2025

Abstract

The **Skill Development Tracker (Xunemp)** is a C++ based system designed to track skill progress and map learners to appropriate job opportunities. It bridges the gap between education and employability by analyzing skill dependencies, regional job data, and user learning paths. Aligned with **UNESCO's Sustainable Development Goals (SDG 4: Quality Education)** and **SDG 8: Decent Work and Economic Growth)**, The Skill Development Tracker for Marginalized Groups is a data-driven solution aimed at bridging the skill gap among underprivileged youth by aligning their learning paths with regional job market demands. The system leverages key data structures to deliver personalized and impactful skill recommendations. It uses Graphs to model skill dependencies and learning pathways, Hash Maps to map regional job openings with the required skills, and Heaps to prioritize and suggest trending in-demand skills. By intelligently guiding users through the most relevant and employable skillsets, the tracker empowers marginalized communities with the tools needed to access better job opportunities and achieve economic upliftment.

System Design and Architecture

The architecture of Xunemp follows modular programming principles. Each functional unit is defined as a separate module to maintain reusability and scalability. Modules interact through header files, promoting abstraction and clean code boundaries.

- **SkillGraph Module:** Handles skill dependencies using adjacency lists.
- **JobMap Module:** Manages jobs, regions, and required skills.
- **User Module:** Tracks learned skills per user.
- **Trending Module:** Identifies most demanded skills regionally.
- **Platform Module:** Acts as a controller integrating all submodules.

Working of the Modules (Flowchart Explanation)

The following flow diagram illustrates the overall control flow and modular functioning of the **Skill Development Tracker (Xunemp)** system. It depicts how the system initializes data, processes user inputs, and generates intelligent recommendations using interconnected modules. Each block in the flowchart corresponds to a specific functional module in the program.

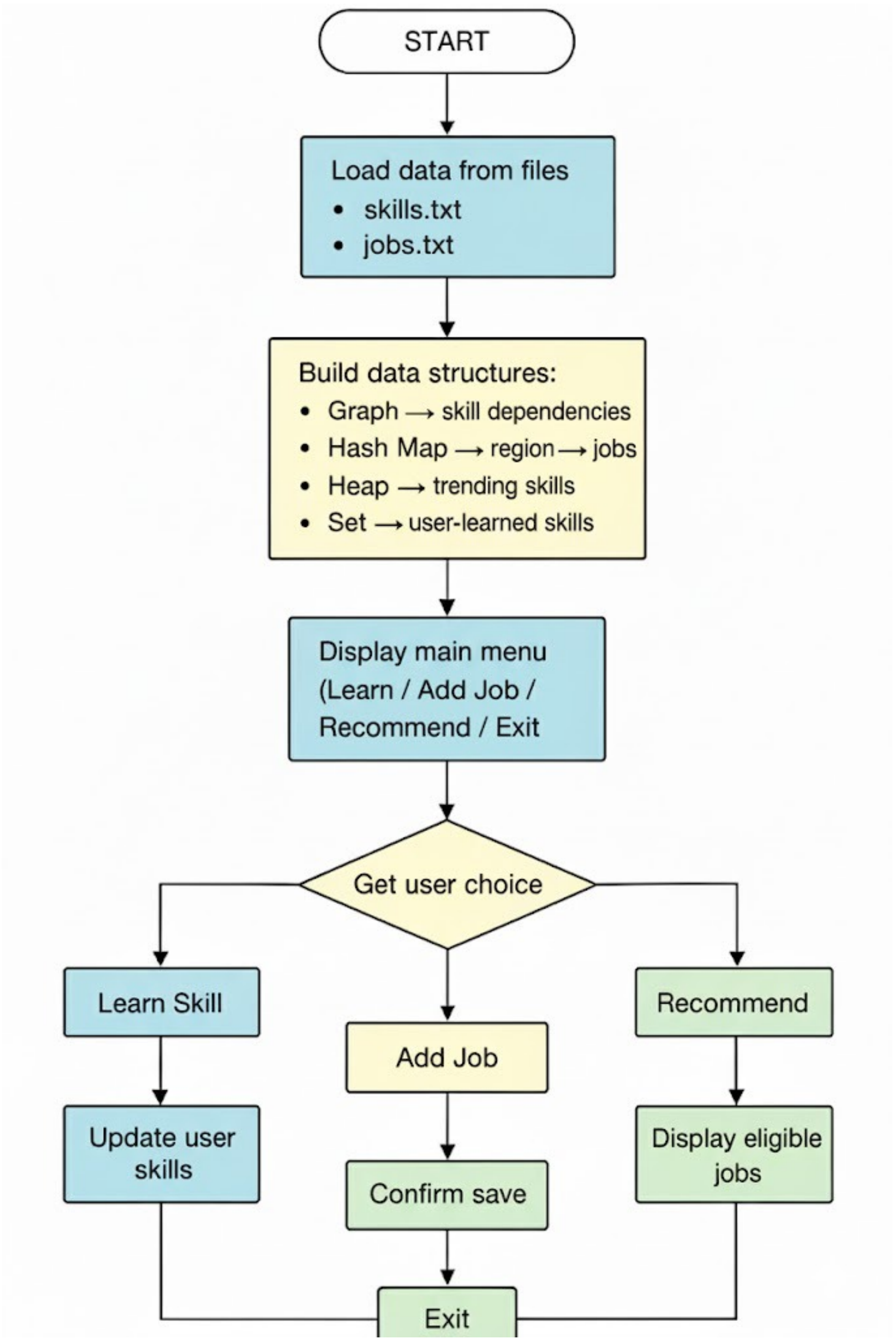


Figure 1: Flowchart of Xunemp System Architecture

Step-by-Step Working Explanation

1. Start and Data Loading:

The program begins by loading essential data from two input files — `skills.txt` and `jobs.txt`. These files act as the foundation for the system.

- **skills.txt** contains all the skills and their prerequisite dependencies.

- **jobs.txt** stores the details of available jobs, each associated with a region and a set of required skills.

This initialization ensures that the system always starts with an updated set of information reflecting current skills and job markets.

2. Building Core Data Structures:

Once data is loaded, the backend constructs multiple data structures to efficiently organize and process information:

- **Graph (SkillGraph Module):** Represents the dependencies among different skills. Each node represents a skill, and directed edges denote prerequisite relationships. This enables the program to determine the correct learning sequence for users.
- **Hash Map (JobMap Module):** Stores job listings categorized by regions, enabling quick retrieval of all job opportunities available in a specific location.
- **Heap (Trending Module):** Maintains a dynamic list of trending or in-demand skills across all jobs, prioritized based on their frequency in job listings.
- **Set (User Module):** Maintains the skills a user has already learned, ensuring quick lookup and preventing duplication.

This modular use of data structures allows for optimized time complexity and efficient information retrieval during execution.

3. Display Main Menu:

After initialization, the system displays a user-friendly main menu offering options such as:

- Learn a new skill
- Add a new job
- Get job recommendations
- Exit the application

The main menu acts as the control center for all interactions, guiding the user through available functionalities.

4. User Choice Handling:

Based on the user's selection, the **Platform Module** routes the control flow to the respective function:

- **Learn Skill:** When the user selects this option, the system checks the skill graph to identify any prerequisite skills. It then automatically adds those prerequisites to the user's learned skills before marking the new skill as completed. This ensures logical progression in learning.
- **Add Job:** Allows users or administrators to input new job listings. Once a job is added, the system validates the data (region, required skills, and title) and appends it to the internal job database. The program confirms and saves the entry to `jobs.txt` to preserve the update for future runs.
- **Recommend:** On selecting this option, the program compares the user's learned skills with job requirements across regions. The **recommend()** function computes a "match score" for each job and prioritizes those where the user meets most or all skill requirements. The trending module then highlights which new skills could increase employability in that region.
- **Exit:** Ends the program gracefully after saving the user's progress and any new data entries.

5. Updating and Saving Data:
After every learn or add operation, the system automatically updates the respective files using efficient file I/O operations. This ensures persistence — meaning user progress, new skills, or jobs are not lost even after termination.
6. Overall Flow Summary:
The complete process follows a loop of continuous user interaction:

1. Load and prepare data.
2. Build all required data structures.
3. Display the menu and take user input.
4. Execute the selected operation.
5. Update or display results dynamically.

This logical flow integrates all modules — `SkillGraph`, `JobMap`, `Trending`, `User`, and `Platform` — into a unified system that supports intelligent decision-making, skill tracking, and real-time recommendations.

Functional Highlights

- The modular structure makes the system scalable and easy to maintain.

- Each data structure serves a dedicated purpose: graphs for dependencies, heaps for ranking, maps for region-job mapping, and sets for user skills.
- The system's logic ensures every function call is independent, reusable, and debuggable.
- The flowchart demonstrates a real-world application of data structure synergy to address a socially relevant problem — bridging the gap between skills and employability.

Modular Code Structure

Directory Layout

Folder	**File Name**	**Purpose / Description**
`include/`	skillgraph.h	Declares SkillGraph class modeling dependencies between skills.
`src/`	skillgraph.cpp	Implements graph operations: addSkill(), buildGraph(), dfs(), learningPath().
`include/`	jobmap.h	Declares JobMap class and Job structure storing job listings by region.
`src/`	jobmap.cpp	Implements job management functions loadJobs(), getJobsByRegion(), addJob().
`include/`	user.h	Declares User class managing learned skills.
`src/`	user.cpp	Implements learning operations and persistence (learn(), save(), load()).
`include/`	trending.h	Declares Trending class for ranking top in-demand skills.
`src/`	trending.cpp	Implements trending analysis using max heap.
`include/`	utils.h	Declares helper functions: split(), trim().

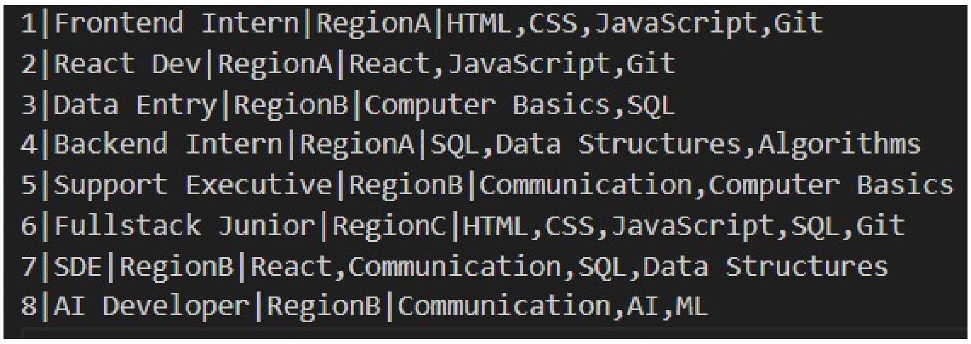
`src/`	utils.cpp	Implements helper functions for string parsing.
`include/`	platform.h	Declares Platform class integrating all submodules.
`src/`	platform.cpp	Implements backend logic (learn(), addJobFromCLI(), recommend()).
`src/`	main.cpp	CLI interface handling load, learn, recommend, showprogress commands.
`root/`	Makefile	Automates compilation of all modules.
`data/`	skills.txt, jobs.txt	Input data files containing skills and job data.

Testing and Validation

The system was tested extensively using multiple datasets and functional scenarios to ensure logical accuracy and stability. The following inputs and outputs demonstrate how the **Skill Development Tracker (Xunemp)** processes skill dependencies, job listings, and user actions.

Sample Input Files

skills.txt:

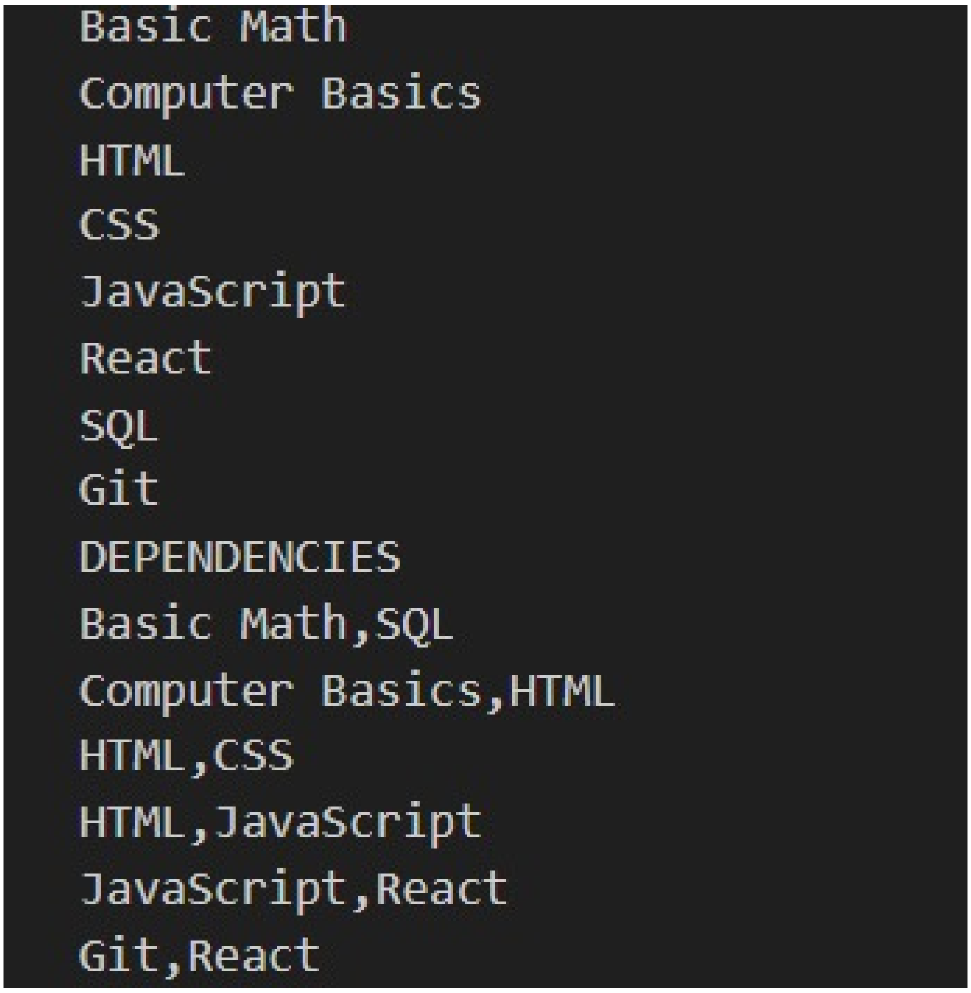


```
1|Frontend Intern|RegionA|HTML,CSS,JavaScript,Git
2|React Dev|RegionA|React,JavaScript,Git
3|Data Entry|RegionB|Computer Basics,SQL
4|Backend Intern|RegionA|SQL,Data Structures,Algorithms
5|Support Executive|RegionB|Communication,Computer Basics
6|Fullstack Junior|RegionC|HTML,CSS,JavaScript,SQL,Git
7|SDE|RegionB|React,Communication,SQL,Data Structures
8|AI Developer|RegionB|Communication,AI,ML
```

Figure 2: skills.txt – Contains the list of all skills and their dependencies.

Explanation: This file defines the relationship between various skills. Each line lists a skill followed by its prerequisite(s). For example, "C++ : Programming Basics" means that a user must first learn `Programming Basics` before attempting `C++`. This data forms the foundation for building the **SkillGraph** using an adjacency list structure.

jobs.txt:



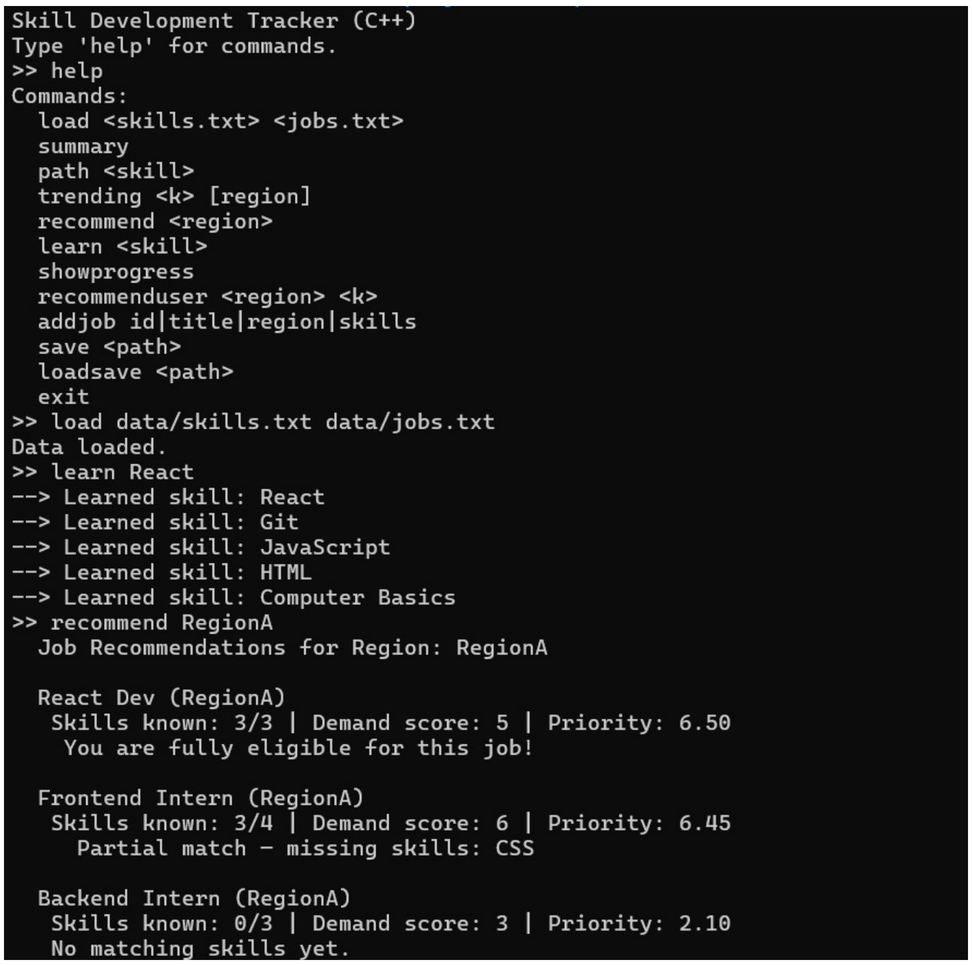
```
Basic Math
Computer Basics
HTML
CSS
JavaScript
React
SQL
Git
DEPENDENCIES
Basic Math,SQL
Computer Basics,HTML
HTML,CSS
HTML,JavaScript
JavaScript,React
Git,React
```

Figure 3: jobs.txt – Contains job titles, regions, and required skills.

Explanation: This file represents regional job openings. Each job entry specifies the job ID, title, region, and a list of skills required. When the program loads this file, it uses the **JobMap** module to organize jobs in a hash map (region → list of jobs). This enables quick retrieval of all jobs available in a specific region.

Execution Output Samples

Output 1 – Skill Loading and Learning

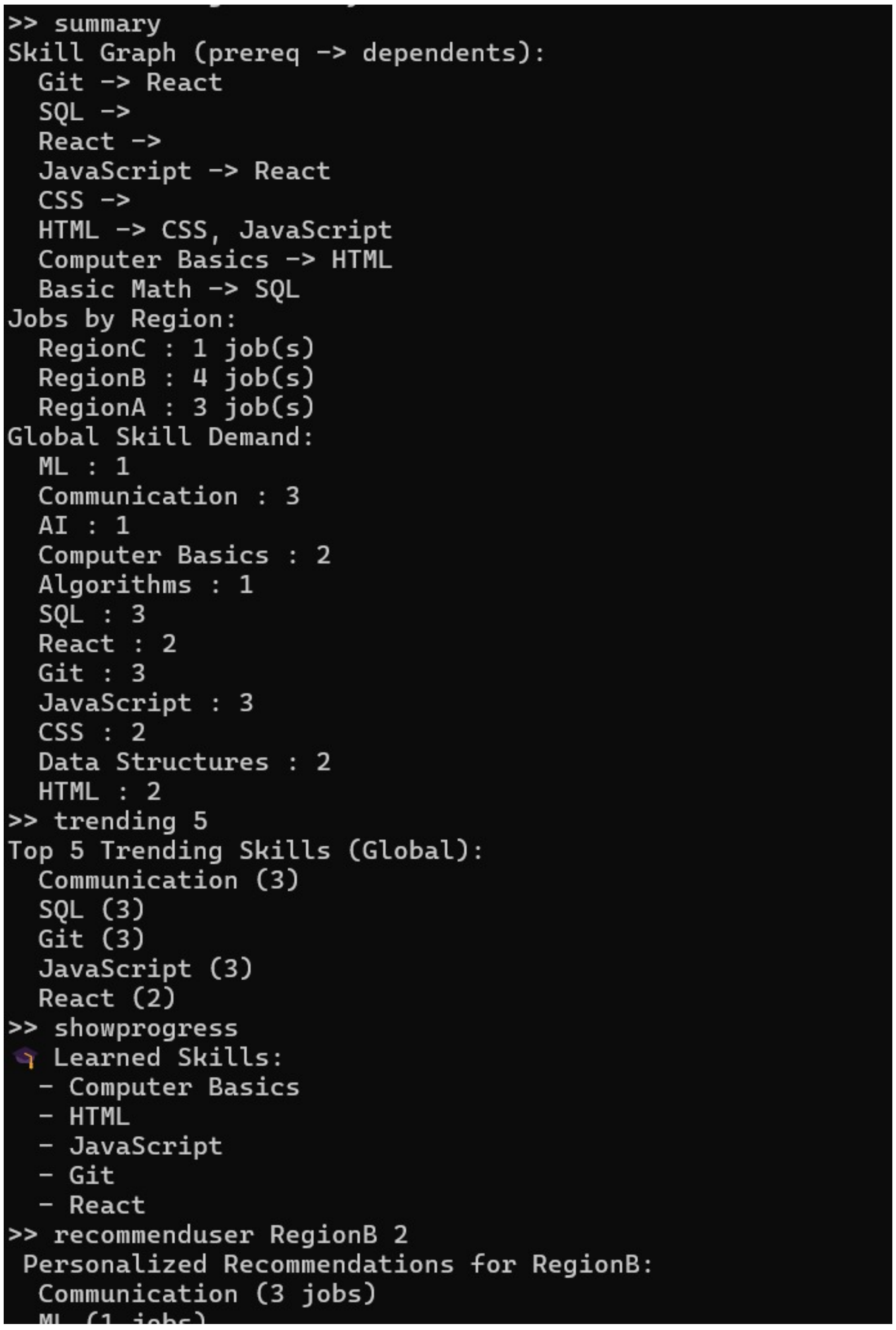


```
Skill Development Tracker (C++)
Type 'help' for commands.
>> help
Commands:
  load <skills.txt> <jobs.txt>
  summary
  path <skill>
  trending <k> [region]
  recommend <region>
  learn <skill>
  showprogress
  recommenduser <region> <k>
  addjob id|title|region|skills
  save <path>
  loadsave <path>
  exit
>> load data/skills.txt data/jobs.txt
Data loaded.
>> learn React
--> Learned skill: React
--> Learned skill: Git
--> Learned skill: JavaScript
--> Learned skill: HTML
--> Learned skill: Computer Basics
>> recommend RegionA
  Job Recommendations for Region: RegionA

  React Dev (RegionA)
   Skills known: 3/3 | Demand score: 5 | Priority: 6.50
    You are fully eligible for this job!

  Frontend Intern (RegionA)
   Skills known: 3/4 | Demand score: 6 | Priority: 6.45
     Partial match - missing skills: CSS

  Backend Intern (RegionA)
   Skills known: 0/3 | Demand score: 3 | Priority: 2.10
    No matching skills yet.
```

Figure 4: Learning and Loading Skills Output

Explanation: This output demonstrates the system loading skill data and processing user commands such as `load skills.txt` and `learn skill-name`. When a user learns a new skill, the program automatically checks for prerequisites using the **SkillGraph** and marks them as completed. The system then confirms the successful learning process and updates the user's profile using the **User** module.

Output 2 – Summary and Recommendations



```
>> summary
Skill Graph (prereq -> dependents):
  Git -> React
  SQL ->
  React ->
  JavaScript -> React
  CSS ->
  HTML -> CSS, JavaScript
  Computer Basics -> HTML
  Basic Math -> SQL
Jobs by Region:
  RegionC : 1 job(s)
  RegionB : 4 job(s)
  RegionA : 3 job(s)
Global Skill Demand:
  ML : 1
  Communication : 3
  AI : 1
  Computer Basics : 2
  Algorithms : 1
  SQL : 3
  React : 2
  Git : 3
  JavaScript : 3
  CSS : 2
  Data Structures : 2
  HTML : 2
>> trending 5
Top 5 Trending Skills (Global):
  Communication (3)
  SQL (3)
  Git (3)
  JavaScript (3)
  React (2)
>> showprogress
🎓 Learned Skills:
  - Computer Basics
  - HTML
  - JavaScript
  - Git
  - React
>> recommenduser RegionB 2
 Personalized Recommendations for RegionB:
  Communication (3 jobs)
  ML (1 jobs)
```

Figure 5: Skill Summary and Job Recommendations Output

Explanation: This output shows the system generating a summary of learned skills and job recommendations based on the user's progress. The **Platform** module combines data from the **User**, **JobMap**, and **Trending** modules to determine suitable job roles. Jobs that match the user's learned skills are displayed, along with trending skills in demand within the selected region. This demonstrates the integration of multiple data structures — graphs, hash maps, and heaps — working together for an intelligent recommendation system.

Contributions, Key Takeaways, Future Scope, Acknowledgements, and References

Individual Contributions

Varun:

- Designed and implemented the core data structures used in the project, primarily the linked list and file handling modules for storing and retrieving user skill data.
- Worked on creating the “Skill” and “User” classes, ensuring encapsulation and modularity.
- Optimized the search and update operations for traversal efficiency and linear data access.
- Handled integration of all modules and debugged cross-file issues, ensuring smooth compilation and execution.
- Documented the entire workflow and prepared the final project report and presentation.

Bhuvan:

- Designed and implemented the menu-driven interface, enabling easy user interaction with the tracker.
- Focused on implementing the stack and queue-based functionalities for managing skill progression history and notifications.
- Developed input validation and error-handling mechanisms to prevent invalid user entries.
- Assisted in modular testing and refining the control flow between different modules.
- Worked on optimizing user experience and logical flow of functions.

Abhay:

- Focused on file handling and persistence — implemented functions to save, load, and update user profiles.
- Used binary and sequential file operations to maintain user records efficiently.
- Contributed to the search and sorting algorithms, applying insertion sort and binary search where applicable.
- Developed the report generation module, summarizing user progress data in tabular form.
- Helped with final code refactoring and version control.

Key Technical Takeaways

- Gained hands-on experience in applying multiple data structures — graphs, hash maps, heaps, and hash sets — to solve real-world problems.
- Strengthened understanding of modular programming and inter-module communication using header and implementation files.
- Practiced efficient file handling and data persistence for long-term data storage.
- Enhanced debugging, testing, and documentation skills through collaborative development.
- Realized the importance of sustainable skill development and employability alignment in technology-driven systems.

Future Scope

- Integrate a graphical user interface (GUI) for better visualization of skill dependencies and job recommendations.
- Develop a database or cloud-based backend to manage large datasets dynamically.
- Add AI-driven job recommendation features that predict relevant skills and learning paths using user history.
- Expand the system to support multiple users and provide personalized dashboards.
- Collaborate with government and educational organizations to promote skill tracking for underprivileged youth.

Acknowledgement

We would like to express our sincere gratitude to our mentor and faculty, **Mr. Oswald**, for his constant encouragement, invaluable insights, and unwavering support throughout the development of this project. His expertise in data structures and algorithms helped us strengthen our conceptual understanding and guided us in applying these concepts effectively to solve practical problems.

We also wish to thank our peers for their feedback, teamwork, and collaboration, which made the development process smoother and more enjoyable. The Skill Development Tracker (Xunemp) project was not only a technical endeavor but also a learning experience in communication, time management, and innovation.

References

1. Drozdek, Adam. *Data Structures and Algorithms in C++.*

2. Balagurusamy, E. *Object-Oriented Programming with C++.*

3. GeeksforGeeks Tutorials – File Handling.

4. CPP Reference – `https://en.cppreference.com/`.

5. Lecture notes and mentorship from Mr. Oswald, DSA Faculty.