SVKM’s NMIMS

Mukesh Patel School of Technology, Management and Engineering

Department of Computer Engineering

B. Tech Computer III Semester B\C division

2023-2024

Subject: Data Structures

Project Title: Mock LinkedIn Implementation

Index:

|  |  |  |
| --- | --- | --- |
| 1. | Chapter 1: Introduction to Project  1.1 Project Description  1.2 Project Application  1.3 Project Scope |  |
| 2. | Chapter 2: Project Functionalities  2.1 Functionalities in detail  2.2 Users of the project |  |
| 3. | Chapter 3: Abstract Data Structure  3.1 Data structures required and its suitability  3.2 ADT of each data structure |  |
| 4. | Chapter 4: Project Implementation  4.1 Code of the project  4.2 Screenshots |  |
| 5. | Chapter 5: Learning, Conclusion and Future Scope  5.1 Your learning  5.2 Your experience  5.3 Conclusion  5.4 Future scope |  |
|  |  |  |

# Chapter 1: Introduction to Project

1.1 Project Description

Mock LinkedIn Implementation:

The "Mock LinkedIn Implementation" is a comprehensive software application designed to provide users with a platform for managing their personal and professional information, connecting with others, and facilitating communication in a structured manner.

User Profiles:

The core element of this system is the concept of user profiles. Users can create, customize, and maintain their profiles, including personal details such as their username, password, position, role, education, skills, and an "about" section where they can share information about themselves. These profiles are organized and stored efficiently in an array, allowing for easy access and management.

Authentication and Login:

To ensure privacy and security, the system offers a robust authentication mechanism. Users must provide their username and password to access their profiles. The login process validates the credentials, ensuring that only authorized users can access their accounts. Additionally, the system enforces a single-user session rule, preventing multiple simultaneous logins with the same account.

Connection Management:

The system allows users to establish and manage connections with other users. This is a fundamental aspect of the platform, enabling users to connect professionally or personally with peers, colleagues, or friends. Connection requests are facilitated through an intuitive process, and pending requests are tracked efficiently.

Messaging System:

Facilitating communication is a key feature of the system. Users can send messages to their connections. Messages are organized and managed through a specialized data structure, the message queue. This queue ensures that messages are delivered in a structured and orderly manner, maintaining the integrity of the communication process.

Profile Deletion:

For users who wish to discontinue their participation on the platform, the system provides an option to delete their profiles. This process involves careful removal of their information, ensuring that data is handled responsibly and securely.

Display All Profiles:

Users have the ability to view information about all the profiles within the system. This feature enhances transparency and encourages interaction among users, fostering a sense of community and networking.

Connection Approval and Disconnection:

Users can manage their connections by approving connection requests from others. They can also disconnect from users with whom they no longer wish to maintain a connection. These actions are executed through a well-structured process, maintaining the user experience.

Pending Connection Requests:

The system allows users to keep track of pending connection requests. Users can review and respond to these requests as part of their interaction with others on the platform.

Logout:

To provide flexibility and security, users can log out of their accounts. This feature is essential for switching between different profiles or safely exiting the system.

The project's design emphasizes user-friendliness and data integrity. It employs a combination of data structures to manage user data, connections, and messages efficiently. The menu-driven interface allows users to interact with the system seamlessly, promoting a positive user experience.

Overall, this project offers a well-structured and feature-rich platform that empowers users to manage their personal and professional relationships, communicate effectively, and tailor their profiles to suit their preferences. It provides the foundation for a social networking system that can be expanded and customized to meet specific requirements and user needs.



1.2 Project Application

The "Mock LinkedIn Implementation" project can have several practical applications, both in professional and personal contexts. Here are some potential applications:

Professional Networking Platform:

The project can serve as a platform for professionals to connect with colleagues, industry peers, and potential employers. Users can build and manage their professional profiles, showcasing their skills and experiences. Job Search and Recruitment:

Job seekers can use the platform to search for job opportunities and connect with recruiters or potential employers. Recruiters can use it to source and connect with potential candidates.

Skill and Knowledge Sharing:

Users can connect with experts in their fields and seek advice, mentorship, or share knowledge and expertise. It can function as a space for professional development.

Entrepreneurial Networking:

Aspiring entrepreneurs can use the platform to find mentors, co-founders, or investors to support their startup ventures. Academic and Educational Networking:

Students, educators, and researchers can connect to share knowledge, collaborate on projects, and engage in academic discussions.

Event Promotion and Networking:

Event organizers can use the platform to promote and manage events, while attendees can connect with others attending the same events.

Online Learning Communities:

The platform can support online learning communities, where users can access educational content, connect with instructors, and collaborate with fellow learners.

1.3 Project Scope

Scope of the project is:

1.3.1 User Profile Creation:

Users can create and customize their profiles with personal and professional information.

1.3.2 Authentication and Login:

Secure user authentication and login system to access their profiles.

1.3.3 Connection Management:

Users can connect with other users and manage these connections.

1.3.4 Messaging System:

Users can send and receive messages to and from their connections.

1.3.5 Profile Deletion:

Users can delete their profiles when needed.

1.3.6 Display All Profiles:

View information about all user profiles in the system.

1.3.7 Connection Approval and Disconnection:

Users can approve or disconnect from their connections.

1.3.8 Pending Connection Requests:

Users can review and respond to pending connection requests.

1.3.9 Logout:

Users can log out of their accounts to switch profiles or exit the system

# Chapter 2: Project Functionalities

2.1 Functionalities in detail User Profile Creation:

* This functionality allows users to create their user profiles by providing various personal and professional information.
* Users can enter details such as their username, password, position, role, education, skills, and an "about" section.
* Profiles are created and stored in the system, providing a platform for users to showcase their background and interests.

Authentication and Login:

* This functionality ensures secure access to user profiles.
* Users must provide their registered username and password to log in.
* The system verifies the credentials, allowing authorized users to access their profiles while protecting against unauthorized access.
* The system enforces a single-user session, preventing multiple simultaneous logins with the same account.

Connection Management:

* This functionality enables users to connect with other users on the platform.
* Users can send connection requests to individuals they wish to connect with, either for professional networking or personal connections.
* Users can also manage their connections, including approving or declining incoming connection requests.

Messaging System:

* This feature facilitates communication between users.
* Users can send and receive messages within the platform.
* Messages are organized and managed through a specialized data structure, such as a message queue.
* This system ensures that messages are delivered in an organized and orderly manner, maintaining the integrity of the communication process.

Profile Deletion:

* Users have the option to delete their profiles if they wish to discontinue their presence on the platform.
* The functionality involves carefully removing the user's information from the system.
* This process is designed to ensure data handling is responsible and secure, preserving user privacy.

Display All Profiles:

* This functionality allows users to view information about all profiles within the system.
* Users can access details of other users' profiles, including their username, position, role, education, skills, and an "about" section.
* It enhances transparency and encourages interaction among users, fostering a sense of community and networking.

Connection Approval and Disconnection:

* Users can manage their connections by approving or declining incoming connection requests.
* This ensures that users have control over their network and the ability to filter their connections based on their preferences.
* Users can also initiate disconnections, severing their connections with other users.

Pending Connection Requests:

* This feature enables users to keep track of pending connection requests from other users.
* Users can review and respond to these requests, deciding whether to accept or decline them.
* It adds a layer of transparency and control to the connection process.

Logout:

* Users can log out of their accounts when needed.
* This feature provides flexibility, allowing users to switch between different profiles or safely exit the system.
* Logging out is essential for ensuring the security of the user's session and account.
* These functionalities collectively provide users with a comprehensive platform for managing their profiles, connecting with others, and facilitating communication in both personal and professional contexts, all within the "Mock LinkedIn Implementation" project.

2.2 Users of the project

Users of the project are:

The "Mock LinkedIn Implementation" project can be valuable to a wide range of potential users in various personal and professional contexts. Here are some potential users of the project:

* Professionals:

Individuals in various industries looking to network with peers, colleagues, and potential employers.

* Job Seekers:

Those actively searching for job opportunities and looking to connect with recruiters and hiring managers.

* Recruiters and Employers:

HR professionals and employers seeking candidates for job openings or networking with other professionals.

* Students:

Students in academic institutions who want to connect with classmates, educators, or potential mentors.

* Educators:

Teachers, professors, and educational institutions interested in connecting with students and colleagues.

* Researchers:

Researchers and academics looking to collaborate with peers in their field.

* Entrepreneurs:

Aspiring entrepreneurs seeking mentors, co-founders, investors, and business connections.

# Chapter 3: Abstract Data Structure

3.1 Data structures required and its suitability

In the "Mock LinkedIn Implementation" project, the following data structures are used, and they are suitable for their respective purposes:

Arrays:

Usage: Arrays are used to store user profiles, pending connection requests, and information about connections.

Suitability: Arrays are suitable for this project as they provide quick and direct access to data elements using indices. User profiles and connection data can be efficiently managed and retrieved using arrays.

Queue:

Usage: A queue is used to manage messages between users in the messaging system.

Suitability: A queue is highly suitable for managing messages in a first-in, firstout (FIFO) manner, ensuring that messages are processed in the order they are received. It maintains the integrity of the message order and provides efficient message delivery.

Graph using Adjacency Matrix:

Usage: An adjacency matrix is used to represent the connections between users in the system.

Suitability: An adjacency matrix is well-suited for modeling and managing user connections in the form of a graph. Each user's connections can be tracked efficiently using this data structure, allowing for quick lookups and updates. It's a practical choice for managing connections.

Structures:

Usage: Structures are used to define the data types for user profiles, messages, and other essential information.

Suitability: Structures are suitable for organizing and grouping related data elements into a single unit. They streamline data management by encapsulating data attributes within distinct structures, making the code more organized and readable.

Pointers:

Usage: Pointers are used for dynamic memory allocation when creating and managing nodes for the message queue.

Suitability: Pointers are suitable for managing memory efficiently, especially in cases where data structures need to grow or shrink dynamically. They ensure optimal memory usage and allocation.

Linked List:

Usage: Linked lists can be used for implementing the message queue efficiently.

Suitability: Linked lists are highly suitable for managing the message queue because they can dynamically grow or shrink as needed. This flexibility ensures that messages are handled in an efficient and organized manner.

These data structures are chosen based on their efficiency, organization, and appropriateness for their respective roles within the "Mock LinkedIn Implementation" project.

3.2 ADT of each data structure

In the "Mock LinkedIn Implementation" project, a variety of data structures are used to efficiently manage different aspects of the system. Here, I'll provide the remaining data structures and their respective Abstract Data Types (ADTs):

\*Arrays:\* c++ ADT Array { attributes:

* string pendingRequests[maxProfiles]

}

\*Queue:\* c++

ADT Queue { attributes:

* Node\* front
* Node\* rear

operations:

* enqueue(sender, receiver, content): Adds an element to the queue.
* dequeue(): Removes an element from the queue.

}

\*Graph (Adjacency Matrix):\* c++ ADT Graph {

attributes:

* bool connections[maxProfiles][maxProfiles] // A 2D array to represent connection status between users

}

\*Structures (UserProfile and Message):\* c++

ADT Structure UserProfile { attributes:

* string username
* string password
* string position
* string role
* string education
* string skill
* string about
* string pendingRequests[maxProfiles]

}

ADT Structure Message { attributes:

* string sender
* string receiver
* string content

}

\*Pointers:\* c++ ADT Pointer { attributes:

* Node\* pointer

}

\*Linked List:\* c++

ADT LinkedList { attributes:

* Node\* head
* Node\* tail

operations:

* insertAtEnd(sender, receiver, content): Inserts a node at the end of the linked list.
* deleteFromFront(): Deletes a node from the front of the linked list.
* display(): Displays elements in the linked list.

}

Chapter 4: Project Implementation Note:

1. code format such as addition of const and unique construction declaration style is due to addition by compiler Clion Used While making of the project
2. project made using ide clion but displayed and documented on vs code as licence of jet brains expired
3. comments added post project presentation for report.

4.1 Code of the project

#include <iostream> #include <string> using namespace std;

class Node;

Node\* front = nullptr;

Node\* rear = nullptr;

class Node { public:

string sender; string receiver; string content;

Node\* next;

Node(const string& sender, const string& receiver, const string& content) : sender(sender), receiver(receiver), content(content), next(nullptr) {}

};

class Queue { public:

void enqueue(const string& sender, const string& receiver, const string& content) {

Node\* newNode = new Node(sender, receiver, content);

if (rear == nullptr) { front = newNode; rear = newNode;

} else {

rear->next = newNode; rear = newNode;

}

}

void dequeue() { if (front == nullptr) {

cout << "QUEUE UNDERFLOW WARNING" << endl;

} else {

Node\* tmp = front; front = front->next; delete tmp;

}

}

void display() {

Node\* tmp2 = front;

while (tmp2 != nullptr) {

cout << "From: " << tmp2->sender << "\n"; cout << "To: " << tmp2->receiver << "\n"; cout << "Message: " << tmp2->content << "\n"; cout << "------\n"; tmp2 = tmp2->next;

}

}

};

// Constants const int maxProfiles = 100;

// Structure to represent a user profile struct UserProfile { string username; string password; string position; string role; string education; string skill; string about;

string pendingRequests[maxProfiles]; // An array to store pending connection requests

};

// Structure to represent a message

struct Message { string sender; string receiver; string content;

};

// Global variables

UserProfile profiles[maxProfiles];

bool connections[maxProfiles][maxProfiles] = {false}; string loggedInUsername = "";

int findProfileIndex(const string& username, int profileCount) { for (int i = 0; i < profileCount; i++) { if (profiles[i].username == username) {

return i;

}

}

return -1; // Profile not found

}

void deleteProfile(const string& username, int& profileCount) { int profileIndex = -1;

for (int i = 0; i < profileCount; i++) { if (profiles[i].username == username) {

profileIndex = i;

break;

}

}

if (profileIndex != -1) {

for (int i = profileIndex; i < profileCount - 1; i++) { profiles[i] = profiles[i + 1];

}

profileCount--;

cout << "Profile '" << username << "' has been deleted.\n";

} else {

cout << "Profile with username '" << username << "' not found.\n";

}

}

void connectWithUser(const string& username, const string& loggedInUsername, int& profileCount) {

int senderIndex = findProfileIndex(loggedInUsername, profileCount); int receiverIndex = findProfileIndex(username, profileCount);

if (senderIndex != -1 && receiverIndex != -1) { UserProfile& receiverProfile = profiles[receiverIndex]; for (int i = 0; i < maxProfiles; i++) {

if (receiverProfile.pendingRequests[i].empty()) { receiverProfile.pendingRequests[i] = loggedInUsername; cout << "Connection Request Sent. \n";

break;

}

}

} else {

cout << "Invalid usernames. Connection request failed.\n";

}

}

void approveConnectionRequest(const string& username, const string& loggedInUsername, int& profileCount) {

int userIndex = findProfileIndex(loggedInUsername, profileCount); int senderIndex = findProfileIndex(username, profileCount);

if (userIndex != -1 && senderIndex != -1) { UserProfile& userProfile = profiles[userIndex]; for (int i = 0; i < maxProfiles; i++) {

if (userProfile.pendingRequests[i] == username) { connections[userIndex][senderIndex] = true; connections[senderIndex][userIndex] = true; cout << "Connection with " << username << " approved.\n"; userProfile.pendingRequests[i] = ""; break;

}

}

} else {

cout << "Invalid usernames. Connection approval failed.\n";

}

}

void disconnectFromUser(const string& username, const string& loggedInUsername, int& profileCount) {

int senderIndex = findProfileIndex(loggedInUsername, profileCount); int receiverIndex = findProfileIndex(username, profileCount);

if (senderIndex != -1 && receiverIndex != -1) { connections[senderIndex][receiverIndex] = false; connections[receiverIndex][senderIndex] = false; cout << "Disconnected from " << username << ".\n";

} else {

cout << "Invalid usernames. Disconnection failed.\n";

}

}

void sendMessage(const string& sender, const string& receiver, Queue& messageQueue) {

string content;

cout << "Enter the message content: "; cin.ignore(); // Clear the newline character getline(cin, content);

messageQueue.enqueue(sender, receiver, content); cout << "Message sent to " << receiver << ".\n";

}

void checkPendingRequests(const string& loggedInUsername, int& profileCount) {

int userIndex = findProfileIndex(loggedInUsername, profileCount);

cout << "Pending connection requests for " << loggedInUsername << ":\n";

UserProfile& userProfile = profiles[userIndex]; for (int i = 0; i < maxProfiles; i++) { if (!userProfile.pendingRequests[i].empty()) {

cout << "Request from: " << userProfile.pendingRequests[i] << "\n"; cout << "1. Accept\n"; cout << "2. Reject\n"; int choice;

cout << "Enter your choice: "; cin >> choice;

if (choice == 1) {

approveConnectionRequest(userProfile.pendingRequests[i], loggedInUsername, profileCount);

} else {

cout << "Connection request from " << userProfile.pendingRequests[i]

<< " rejected.\n";

userProfile.pendingRequests[i] = "";

}

}

}

}

void viewAndReplyMessages(const string& loggedInUsername, int& profileCount, Queue& messageQueue) {

int userIndex = findProfileIndex(loggedInUsername, profileCount); cout << "Messages for " << loggedInUsername << ":\n"; messageQueue.display();

cout << "Enter username of the sender to reply to (or enter 0 to return): "; string senderToReply; cin >> senderToReply;

if (senderToReply == "0") { return;

}

int senderIndex = findProfileIndex(senderToReply, profileCount); if (senderIndex == -1) { cout << "User not found.\n"; return;

}

sendMessage(loggedInUsername, senderToReply, messageQueue);

}

int main() {

cout << "User Profile Creation and Management System\n"; char choice;

bool isLoggedIn = false;

int profileCount = 0; // Local variable for profile count

Queue messageQueue; // Initialize the message queue

do {

if (isLoggedIn) {

cout << "Logged in as " << loggedInUsername << ".\n"; cout << "1. Connect with another user\n"; cout << "2. Disconnect from another user\n"; cout << "3. Send a message\n";

cout << "4. Check pending connection requests\n"; cout << "7. View and reply to messages\n"; cout << "8. Logout\n"; cout << "Enter your choice: "; cin >> choice; string usernameToConnect; string usernameToDisconnect; switch (choice) {

case '1': // Connect with another user cout << "Enter the username of the user you want to connect with:

";

cin >> usernameToConnect;

connectWithUser(usernameToConnect, loggedInUsername, profileCount);

break;

case '2': // Disconnect from another user

cout << "Enter the username of the user you want to disconnect from: ";

cin >> usernameToDisconnect;

disconnectFromUser(usernameToDisconnect, loggedInUsername, profileCount);

break;

case '3': // Send a message

cout << "Enter the username of the user you want to message: "; cin >> usernameToConnect;

sendMessage(loggedInUsername, usernameToConnect, messageQueue);

break;

case '4': // Check pending connection requests checkPendingRequests(loggedInUsername, profileCount); break;

case '7': // View and reply to messages

viewAndReplyMessages(loggedInUsername, profileCount, messageQueue);

break;

case '8': // Logout isLoggedIn = false; loggedInUsername = ""; cout << "Logged out successfully.\n"; break;

default:

cout << "Invalid choice. Please try again.\n";

}

} else {

cout << "1. Create a new profile\n"; cout << "2. Delete a profile\n"; cout << "3. Login\n"; cout << "4. Display all profiles\n"; cout << "6. Exit\n"; cout << "Enter your choice: "; cin >> choice;

switch (choice) { case '1': // Create a new profile if (profileCount < maxProfiles) {

UserProfile newUserProfile;

cout << "Enter your username: "; cin >> newUserProfile.username; cout << "Enter your password: "; cin >> newUserProfile.password; cout << "Enter your position: "; cin >> newUserProfile.position; cout << "Enter your role: "; cin >> newUserProfile.role; cout << "Enter your education: "; cin >> newUserProfile.education; cout << "Enter your skills: "; cin >> newUserProfile.skill;

cin.ignore(); // Consume the newline character cout << "Tell us about yourself: "; getline(cin, newUserProfile.about);

profiles[profileCount] = newUserProfile; profileCount++;

cout << "Profile created successfully!\n";

} else {

cout << "Maximum number of profiles reached. Cannot create more.\n";

} break;

case '2': // Delete a profile if (profileCount > 0) {

cout << "Enter the username of the profile you want to delete: "; string usernameToDelete; cin >> usernameToDelete;

deleteProfile(usernameToDelete, profileCount);

} else {

cout << "No profiles to delete.\n";

} break;

case '3': // Login if (!isLoggedIn) {

string username, password; cout << "Enter your username: "; cin >> username;

cout << "Enter your password: "; cin >> password;

int index = findProfileIndex(username, profileCount); if (index != -1 && profiles[index].password == password) { isLoggedIn = true;

loggedInUsername = username;

cout << "Logged in as " << username << ".\n";

} else {

cout << "Invalid username or password.\n";

}

} else {

cout << "You are already logged in as " << loggedInUsername << ". Please logout first.\n";

} break;

case '4': // Display all profiles cout << "All created profiles:\n"; if (profileCount == 0) {

cout << "No profiles to display.\n";

} else {

for (int i = 0; i < profileCount; i++) { cout << "Profile " << (i + 1) << ":\n";

cout << "Username: " << profiles[i].username << "\n"; cout << "Position: " << profiles[i].position << "\n"; cout << "Role: " << profiles[i].role << "\n"; cout << "Education: " << profiles[i].education << "\n"; cout << "Skills: " << profiles[i].skill << "\n"; cout << "About: " << profiles[i].about << "\n\n";

} } break;

case '6': // Exit

cout << "Exiting the program.\n"; break;

default:

cout << "Invalid choice. Please try again.\n";

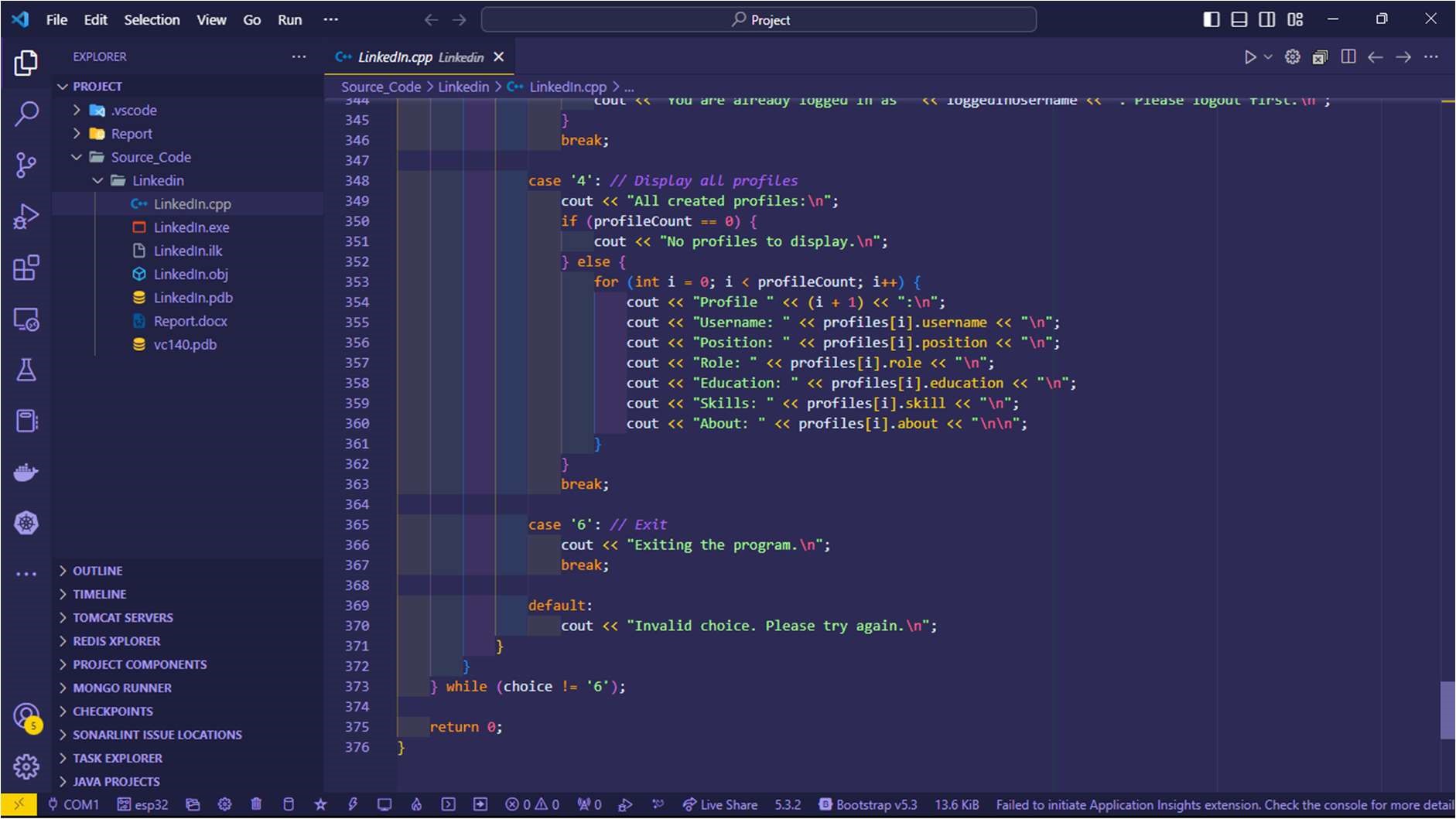
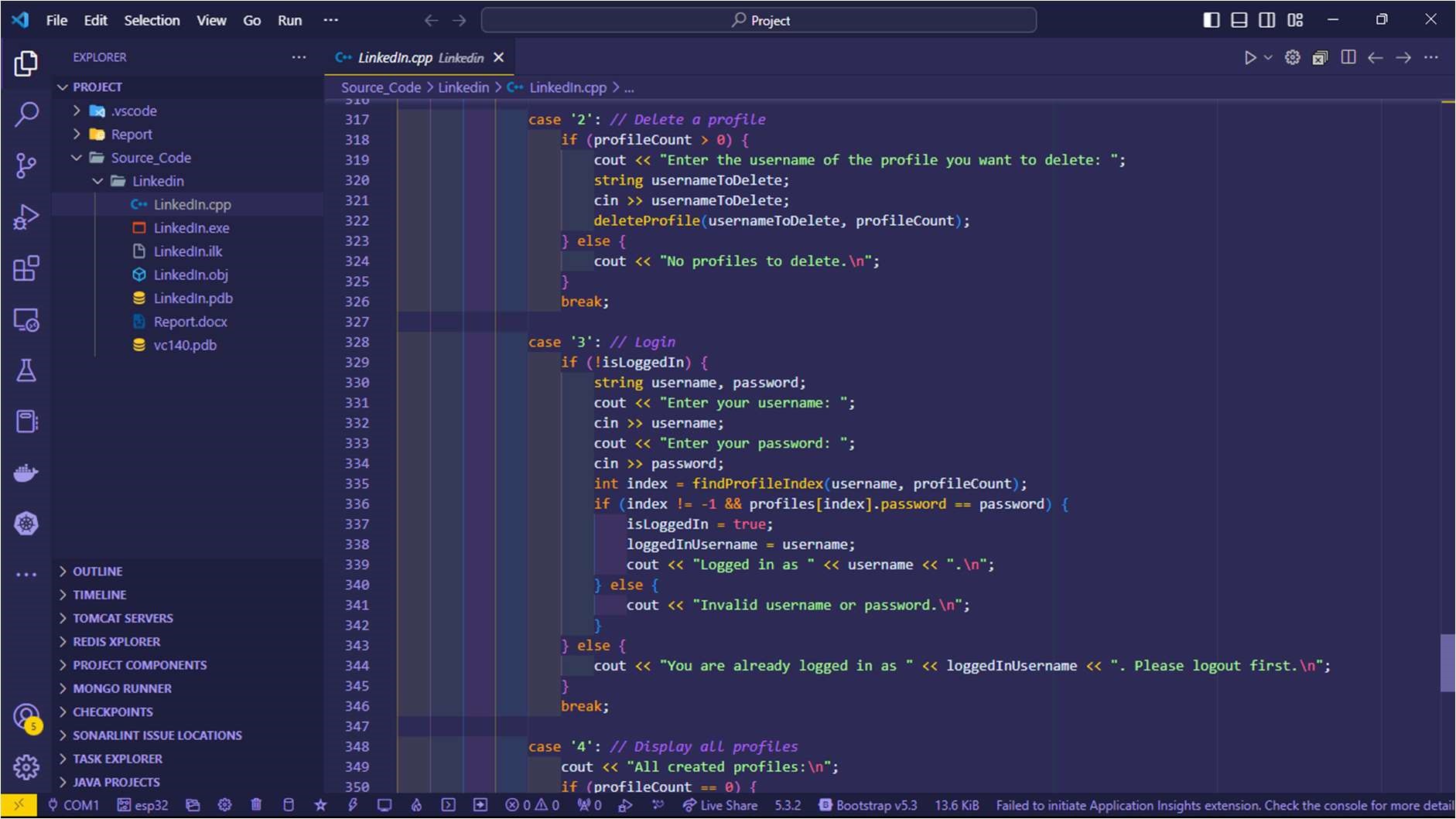
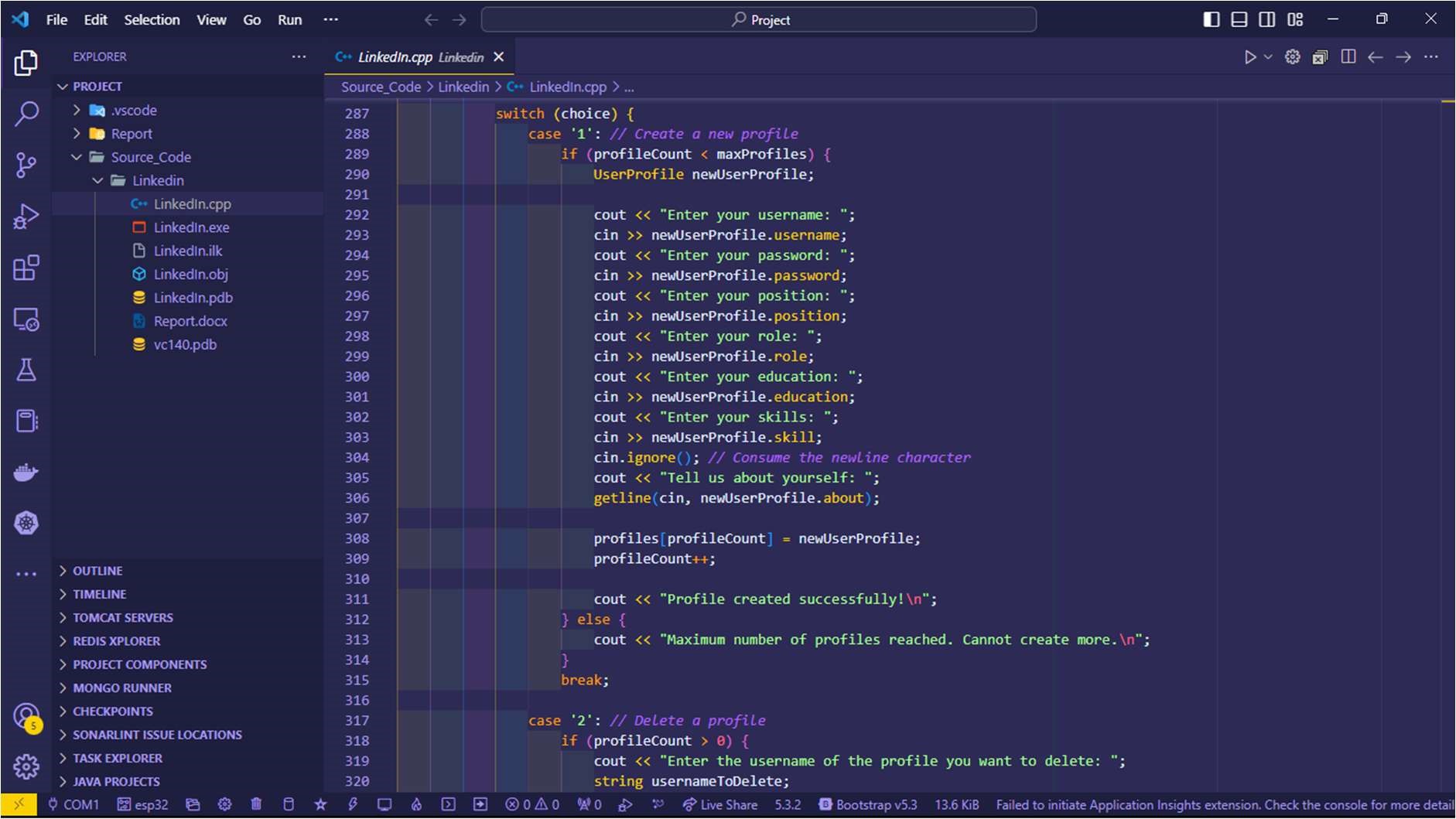
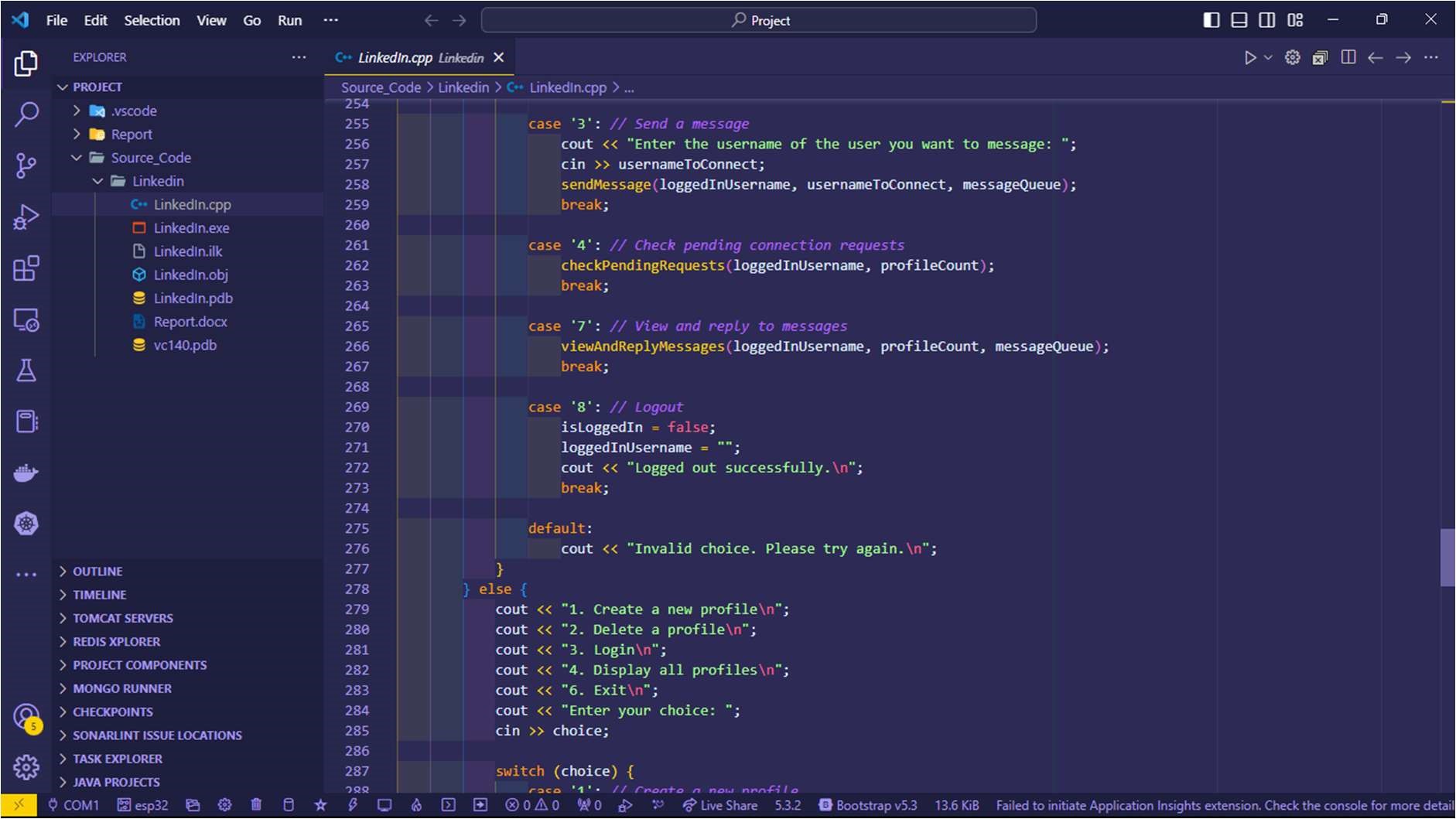
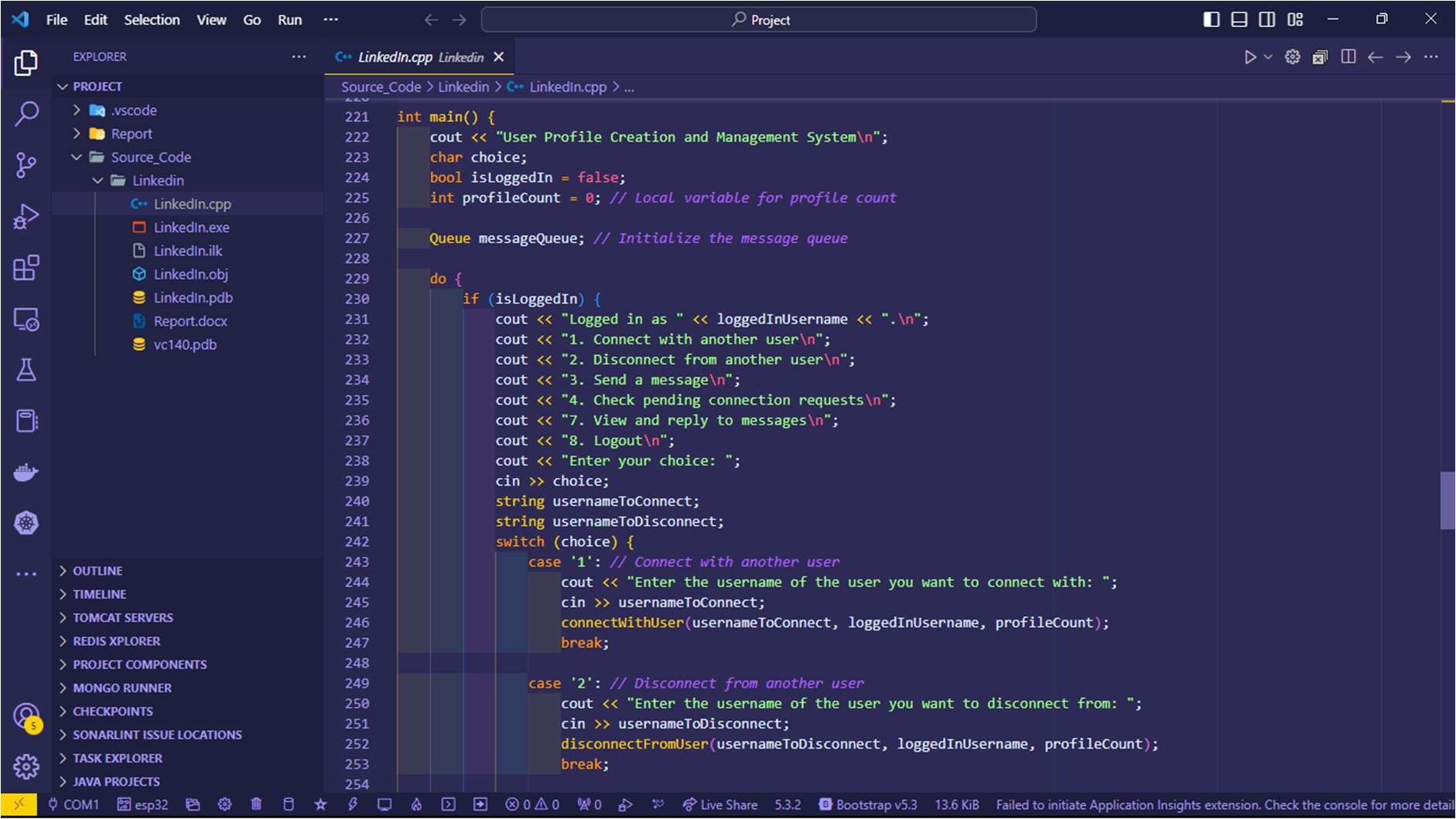
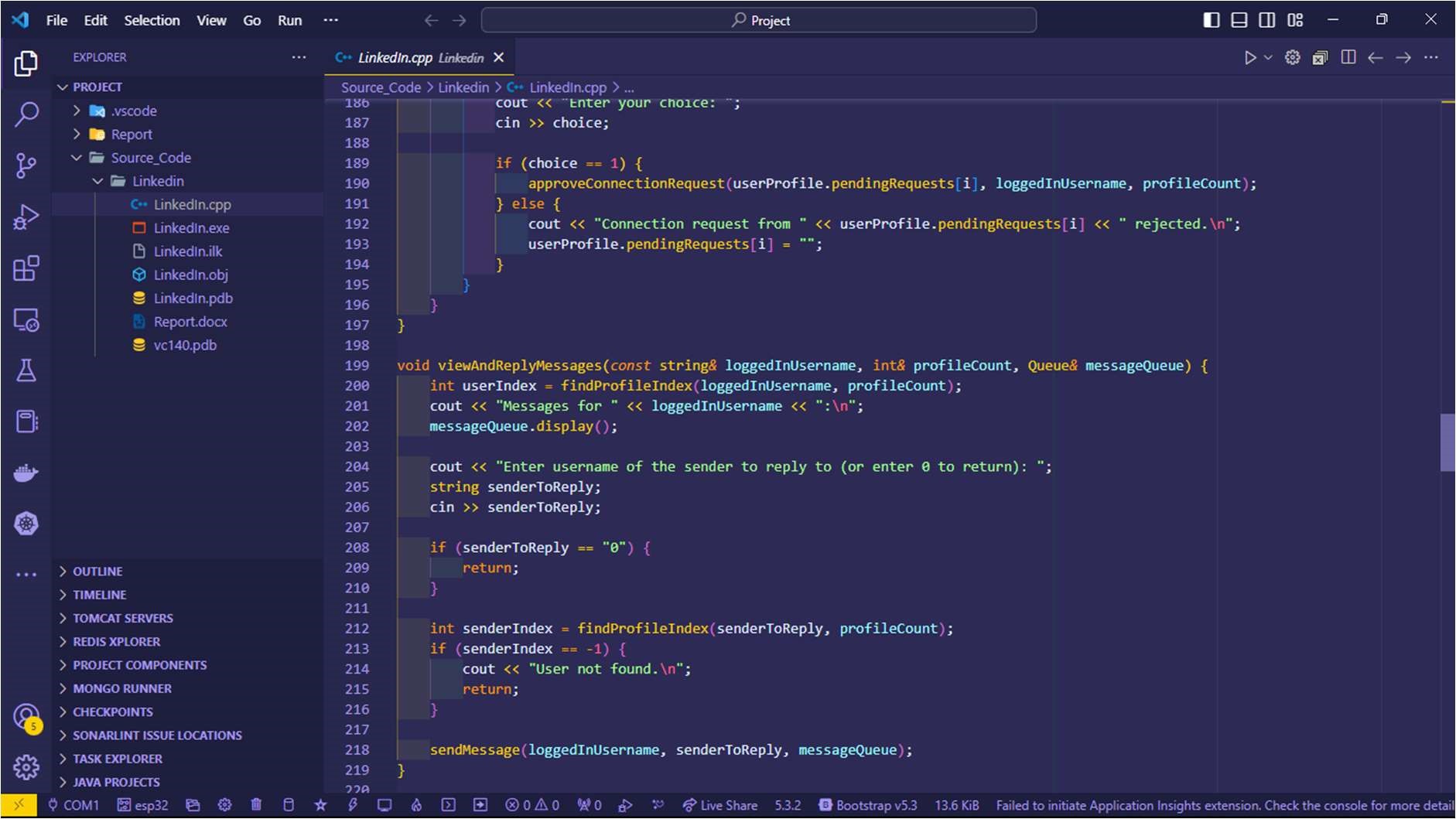
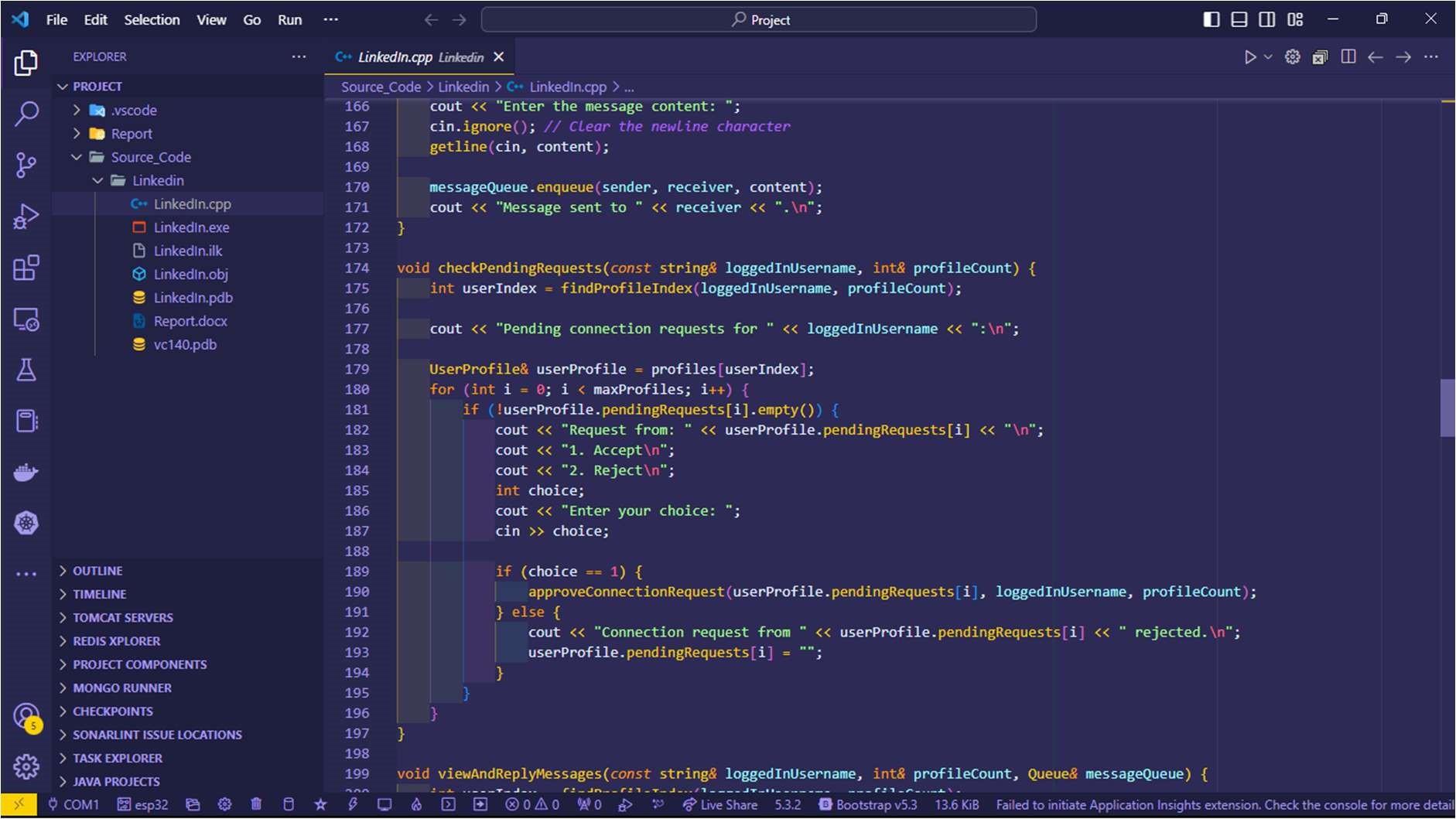
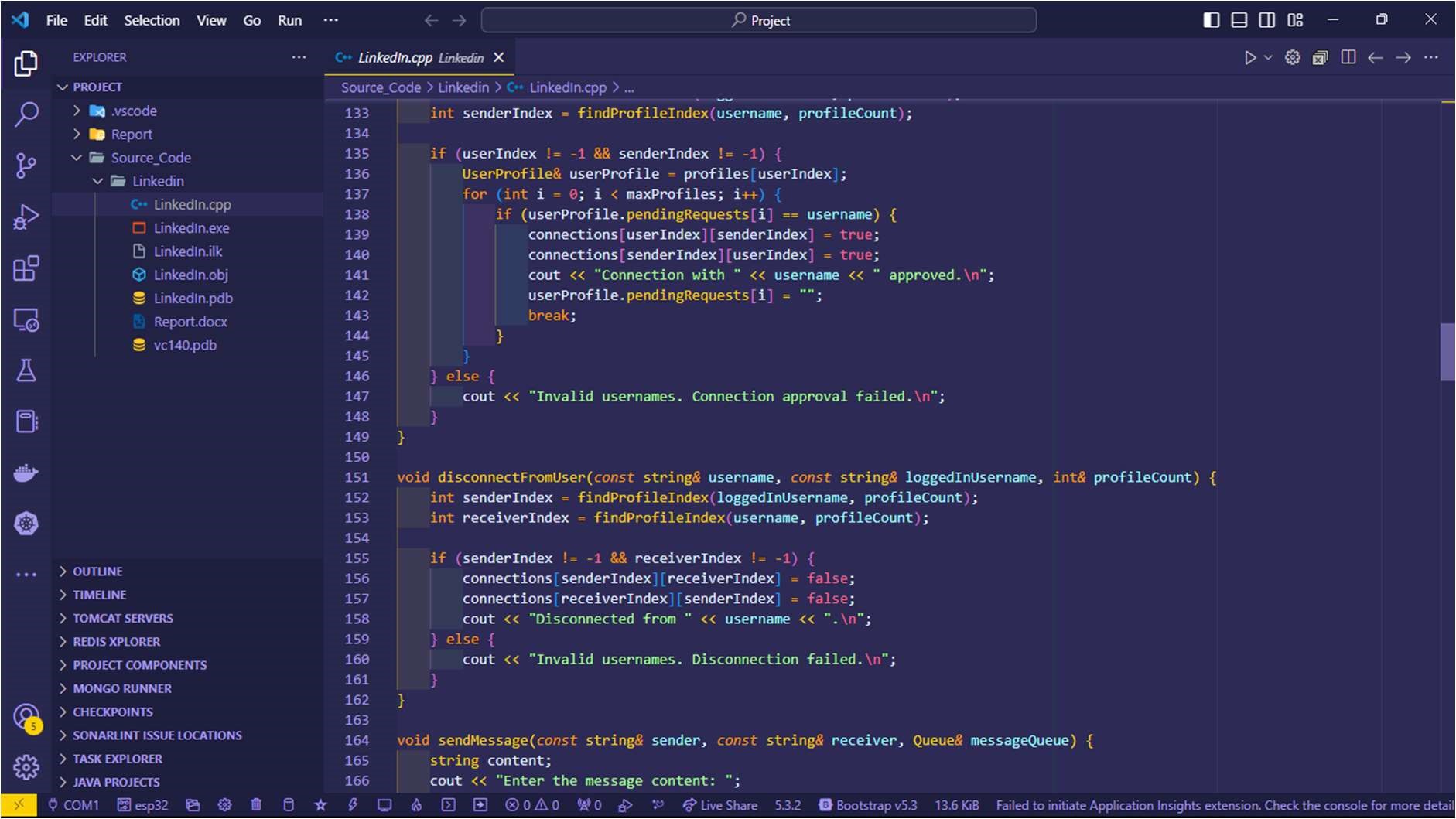
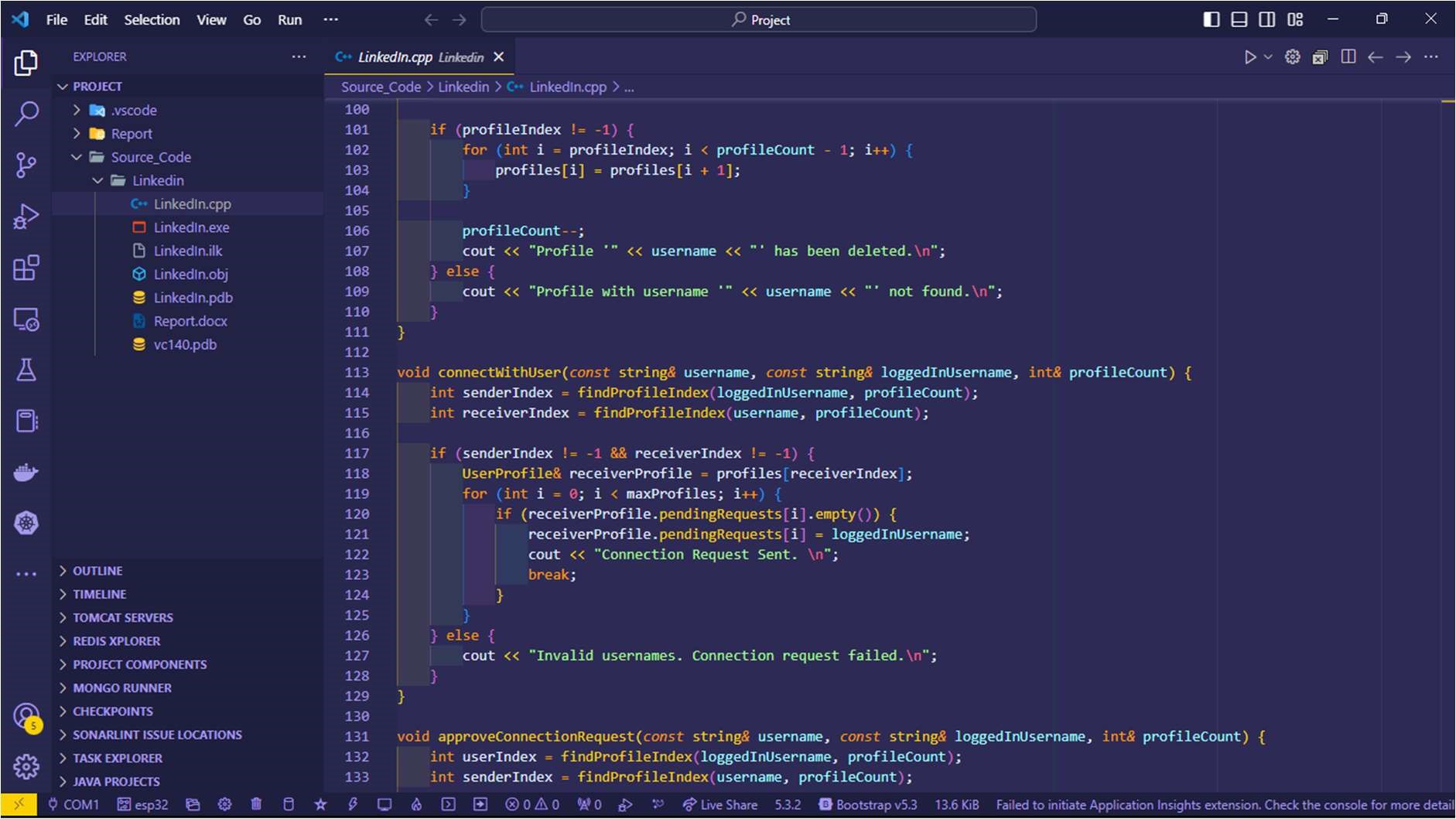
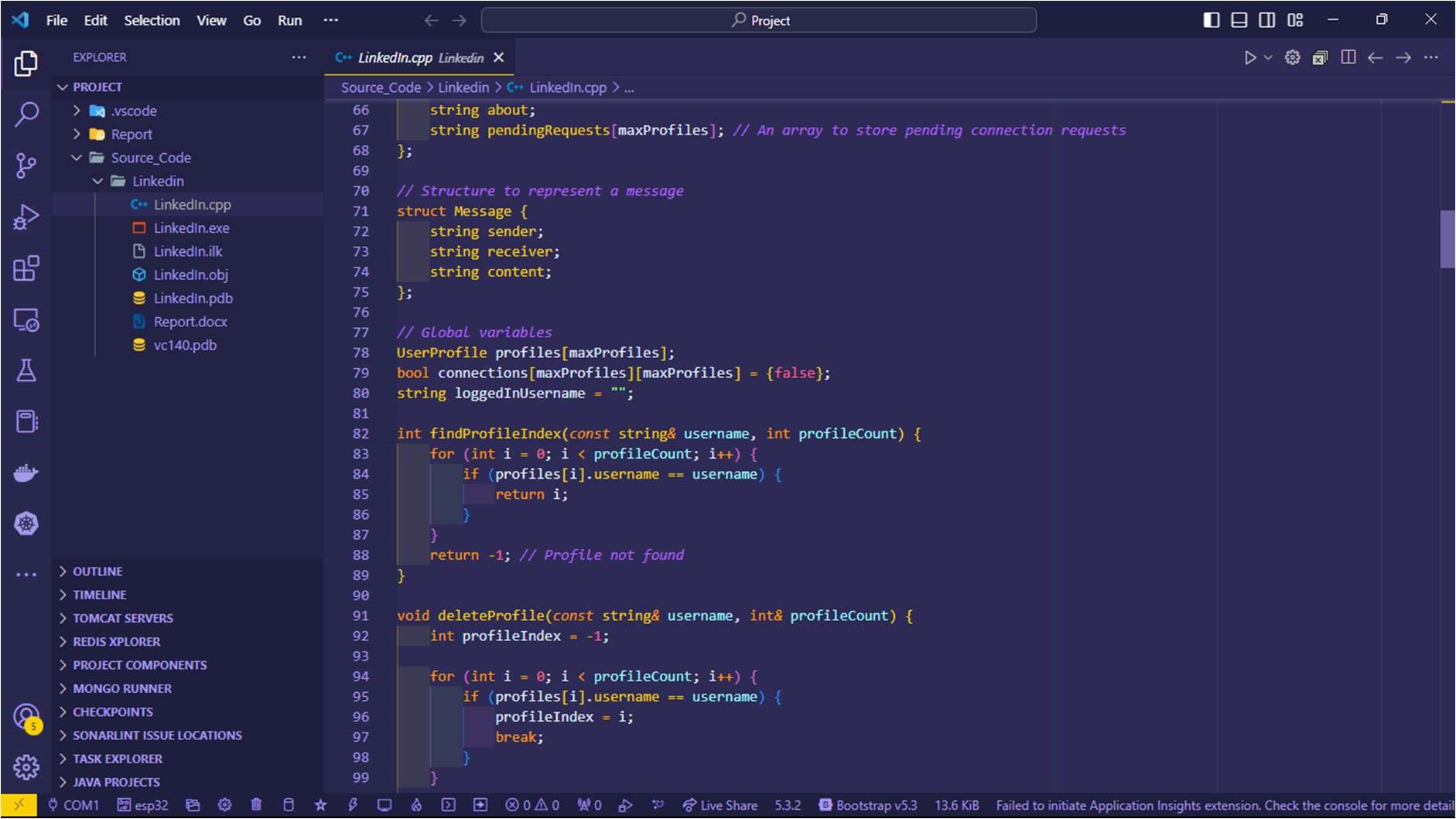
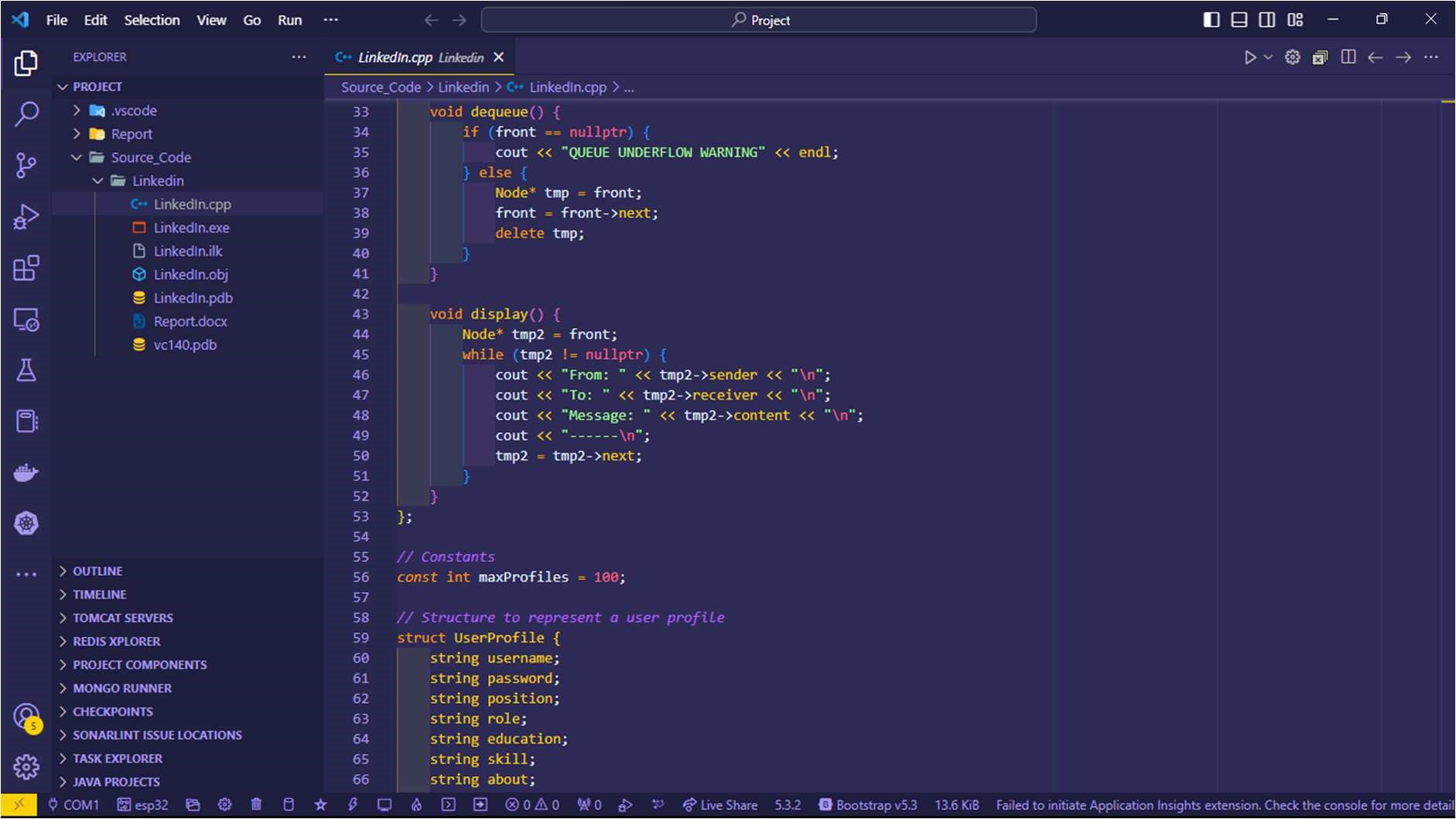
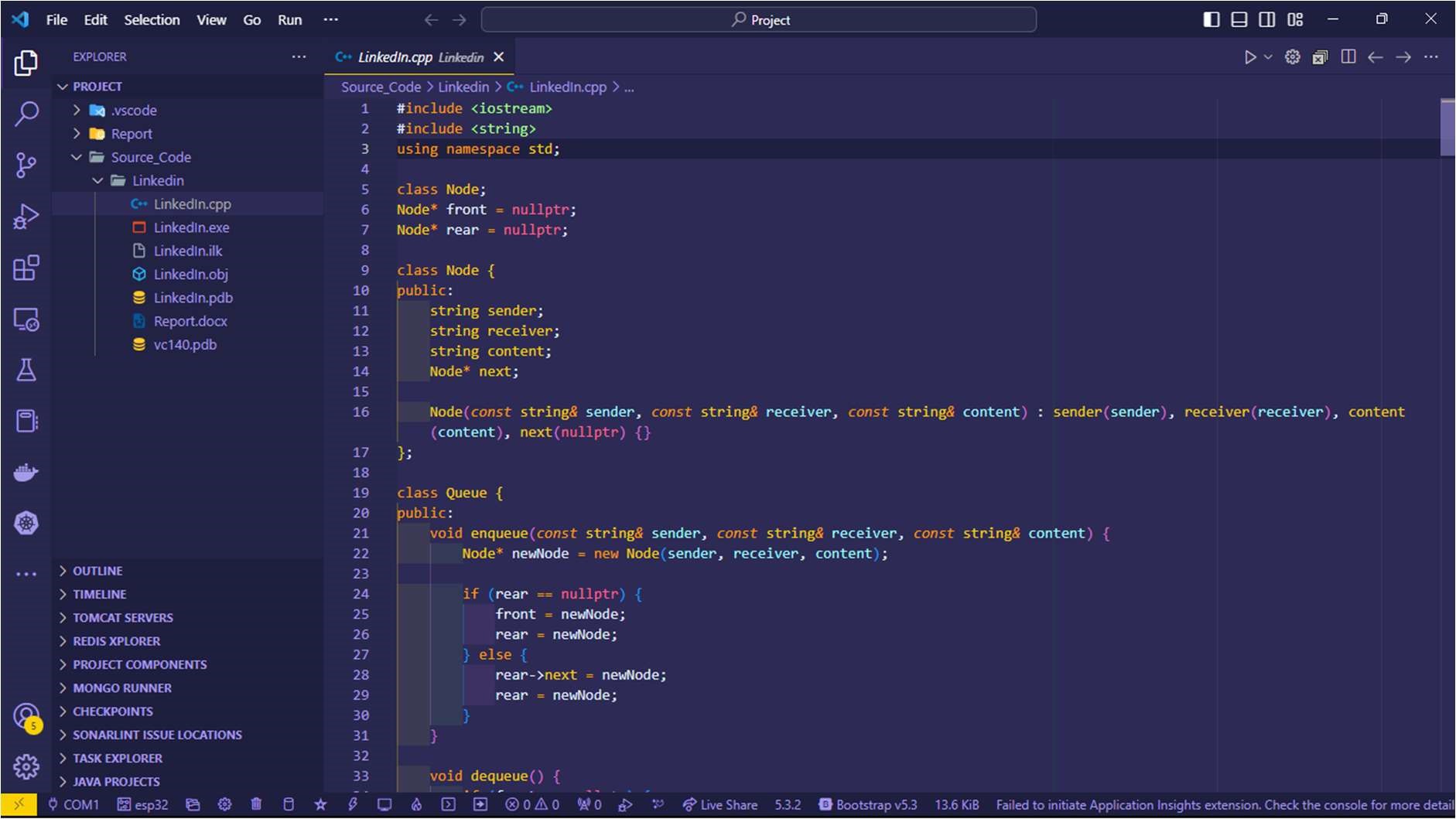
}

}

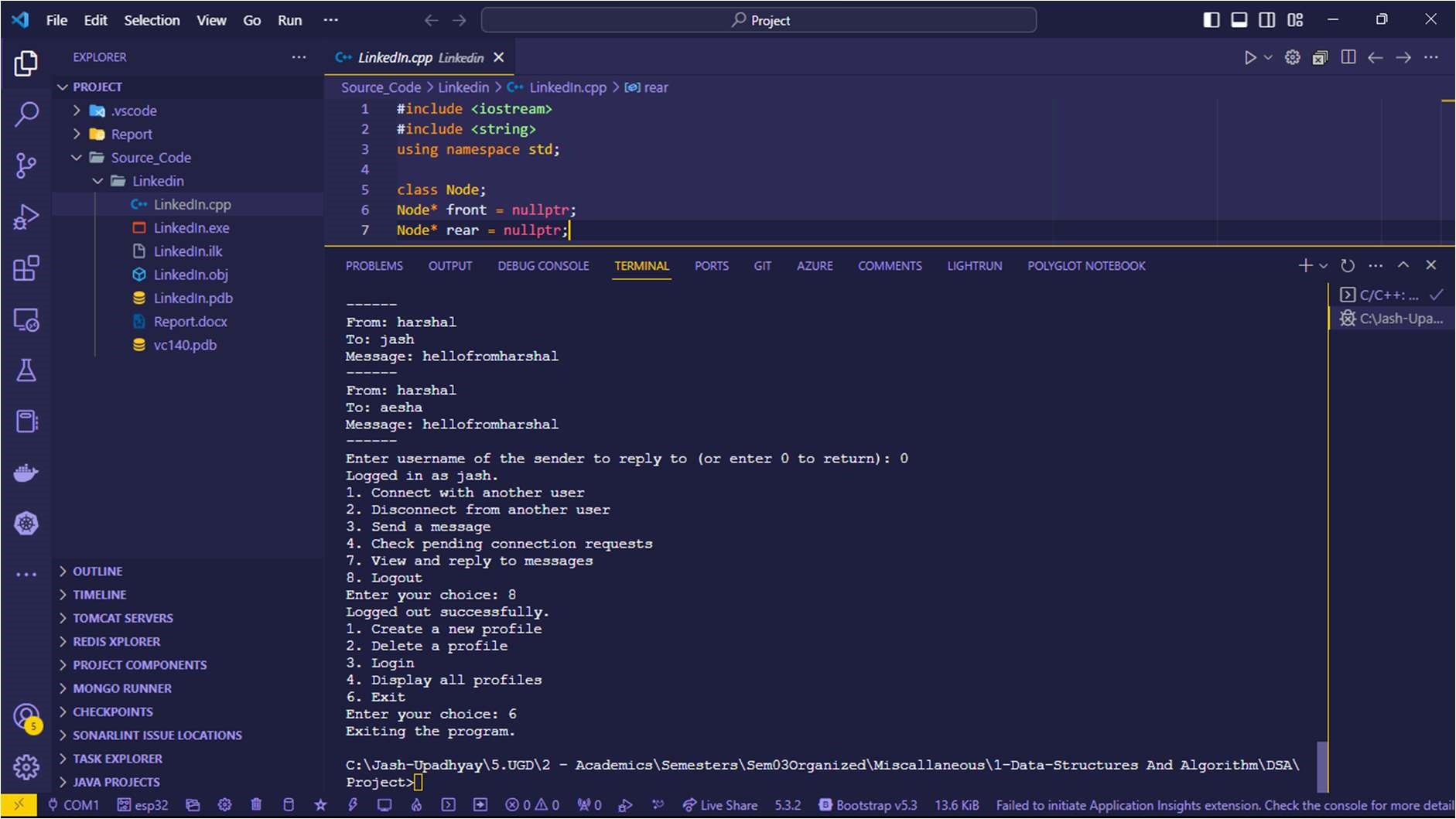
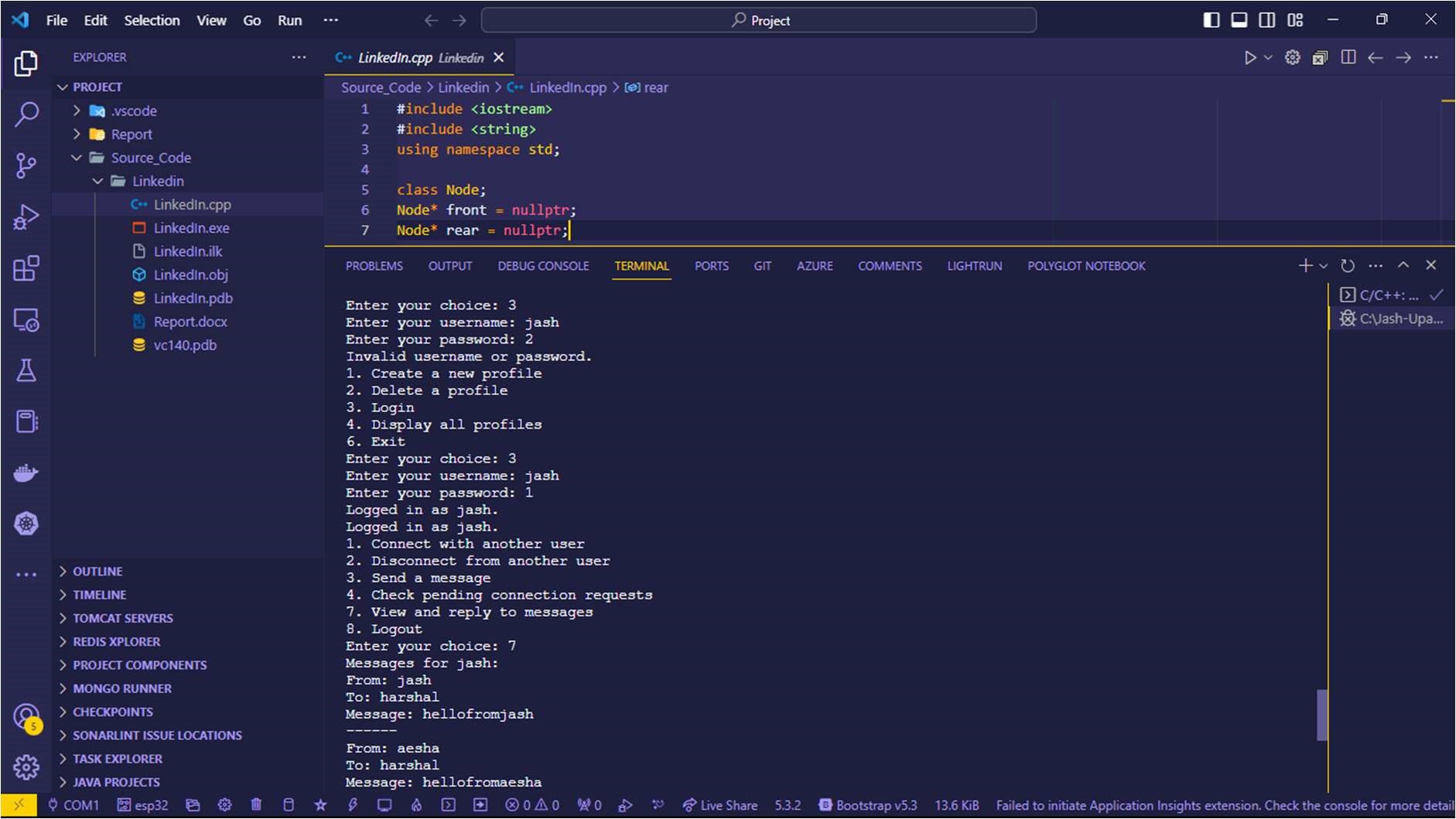
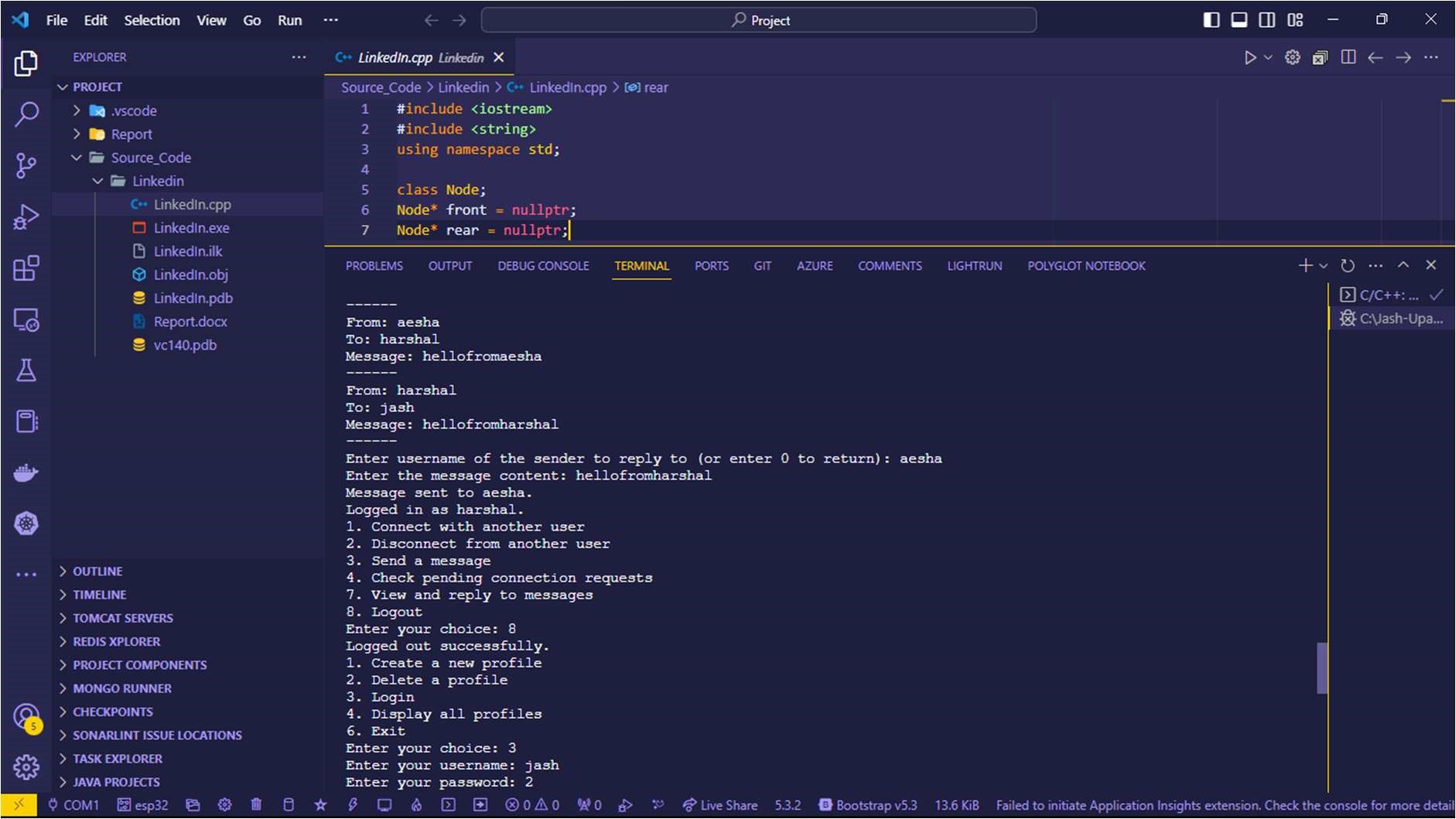
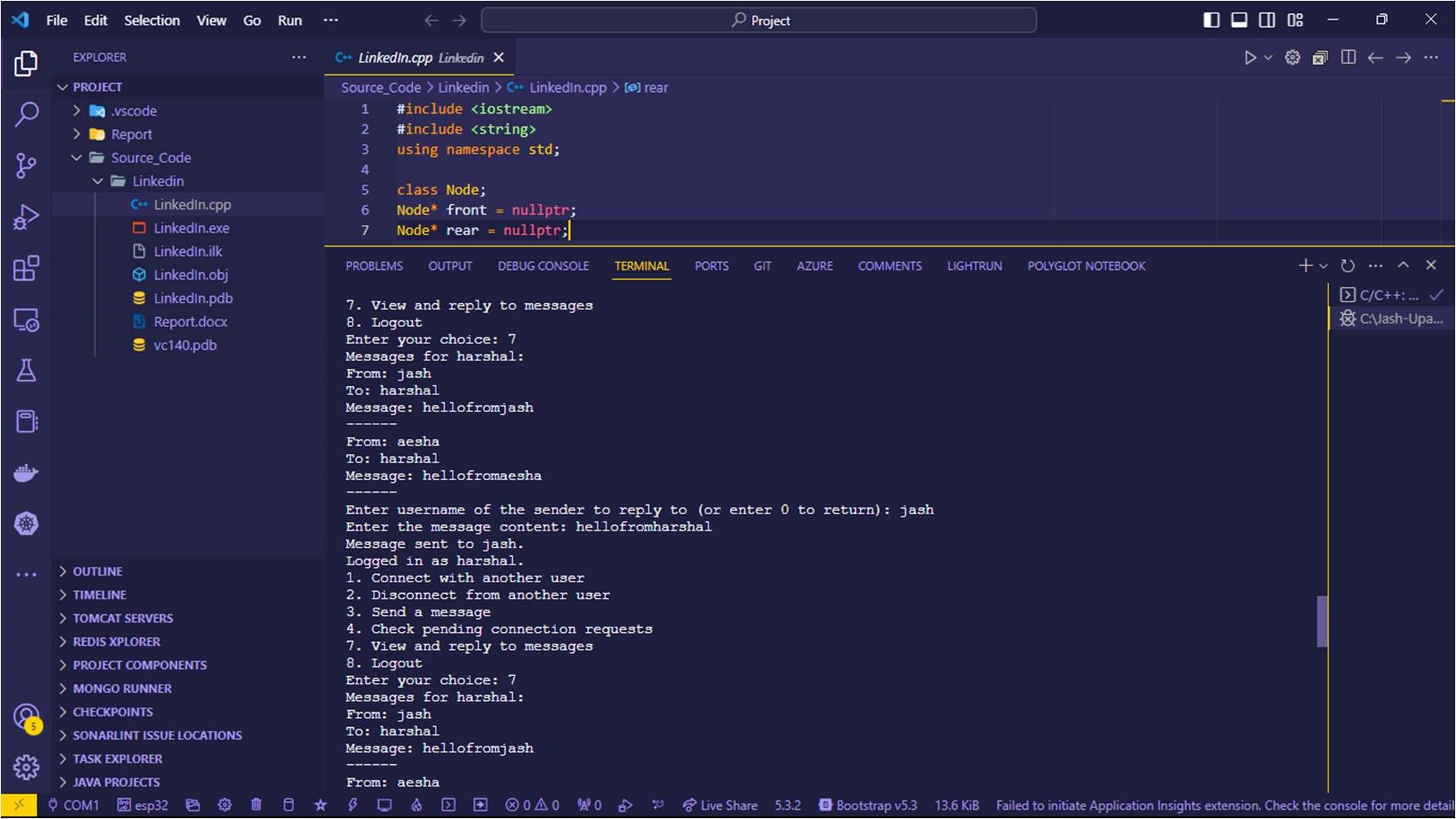
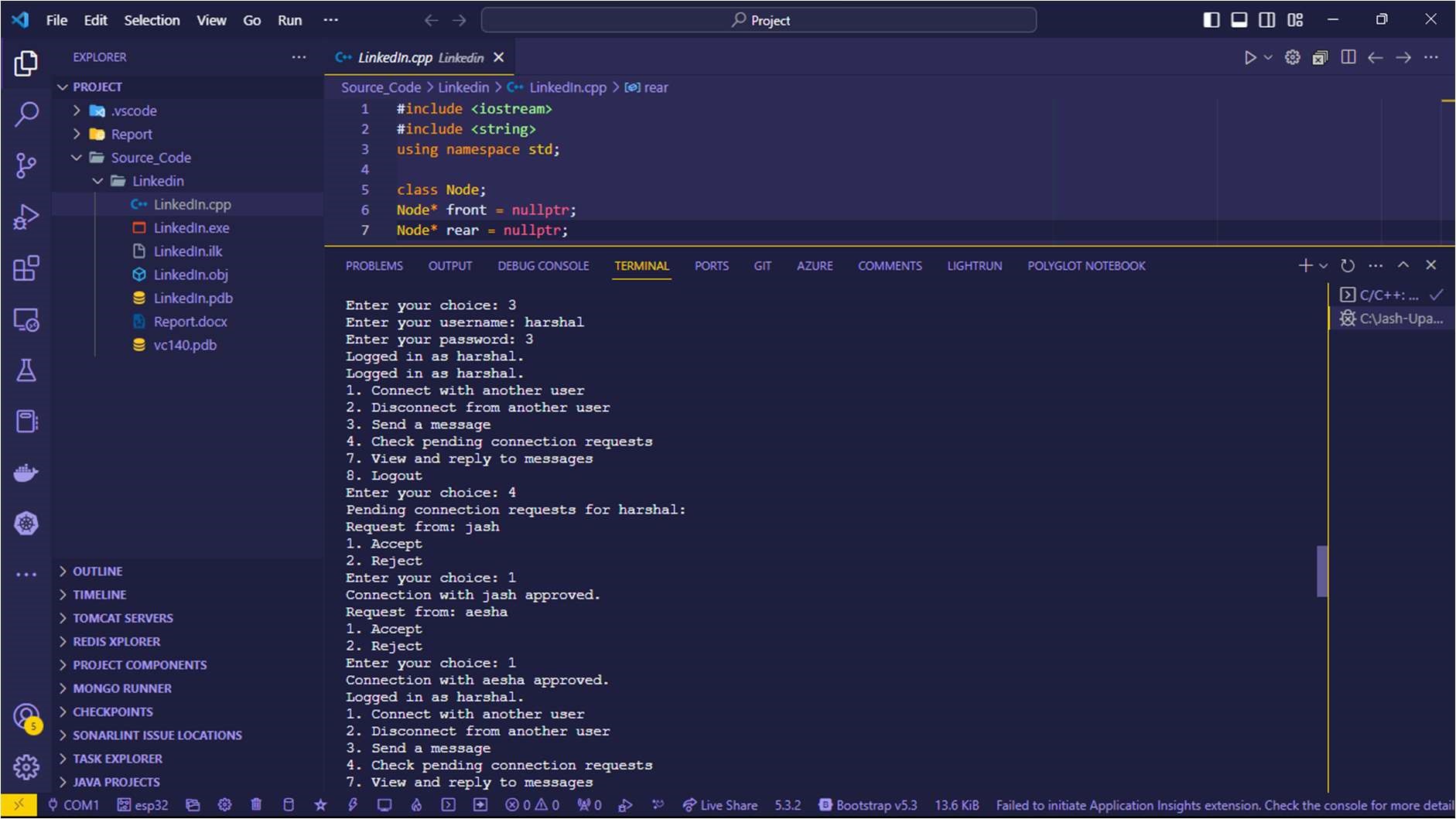
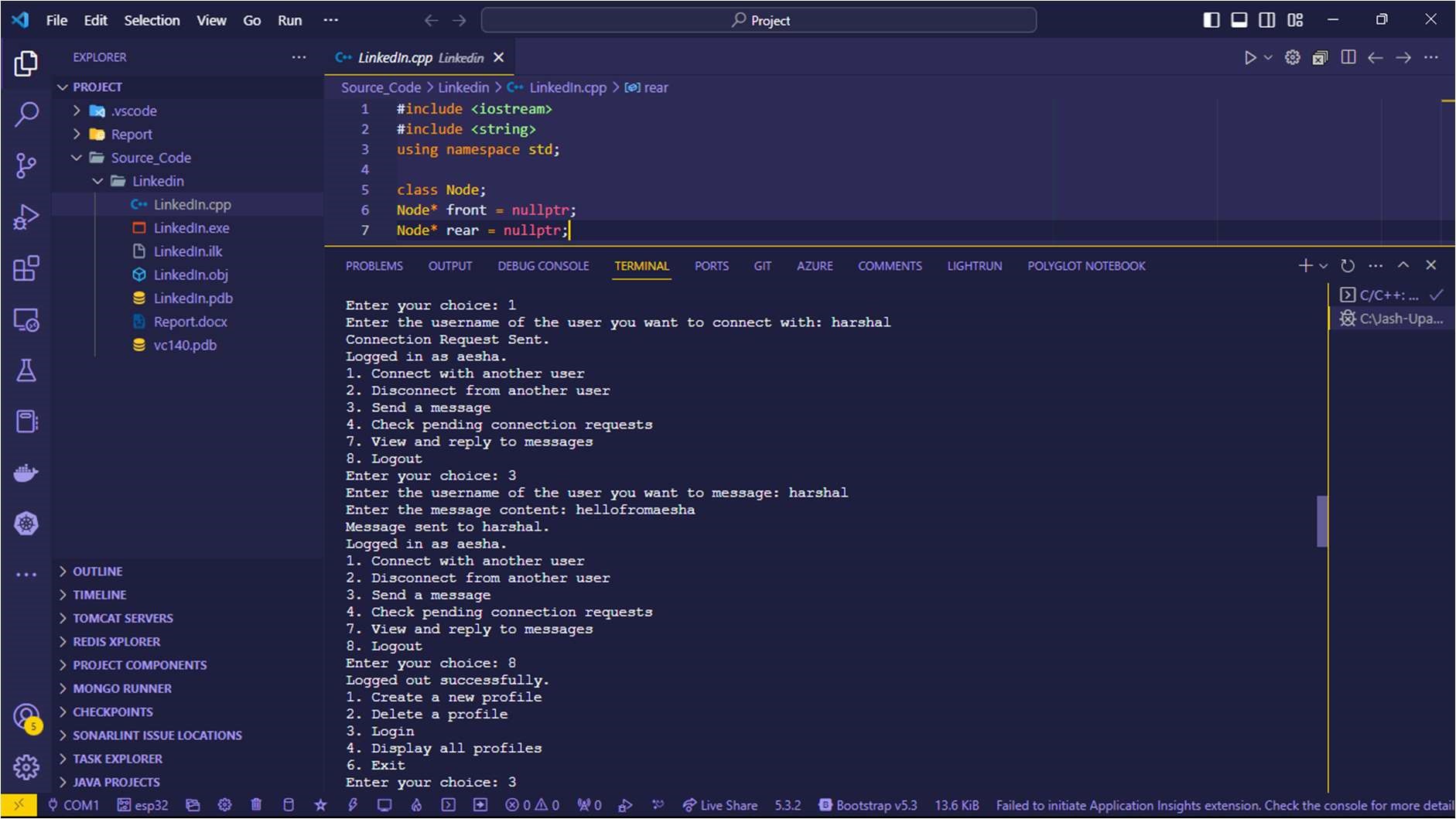
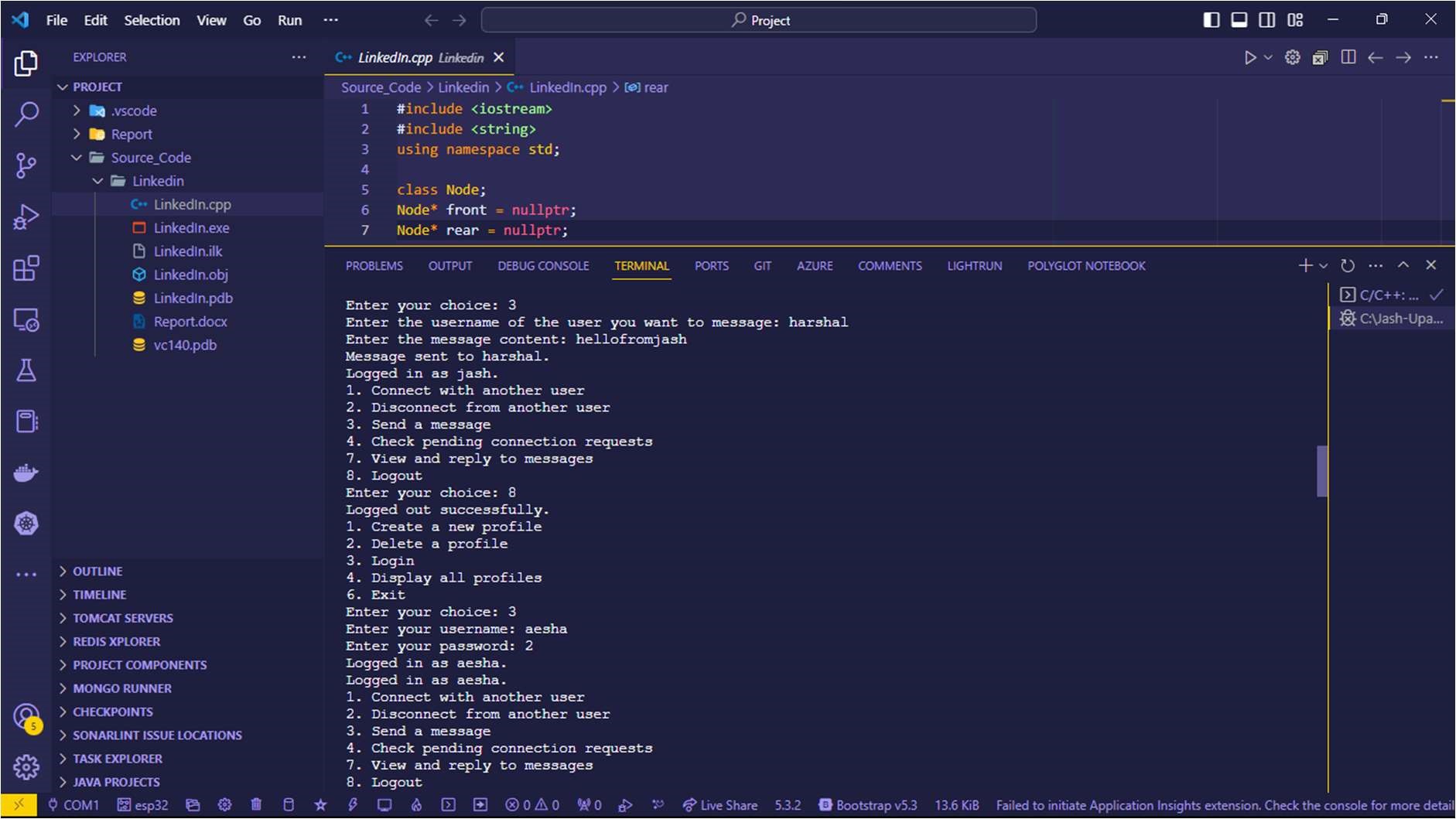
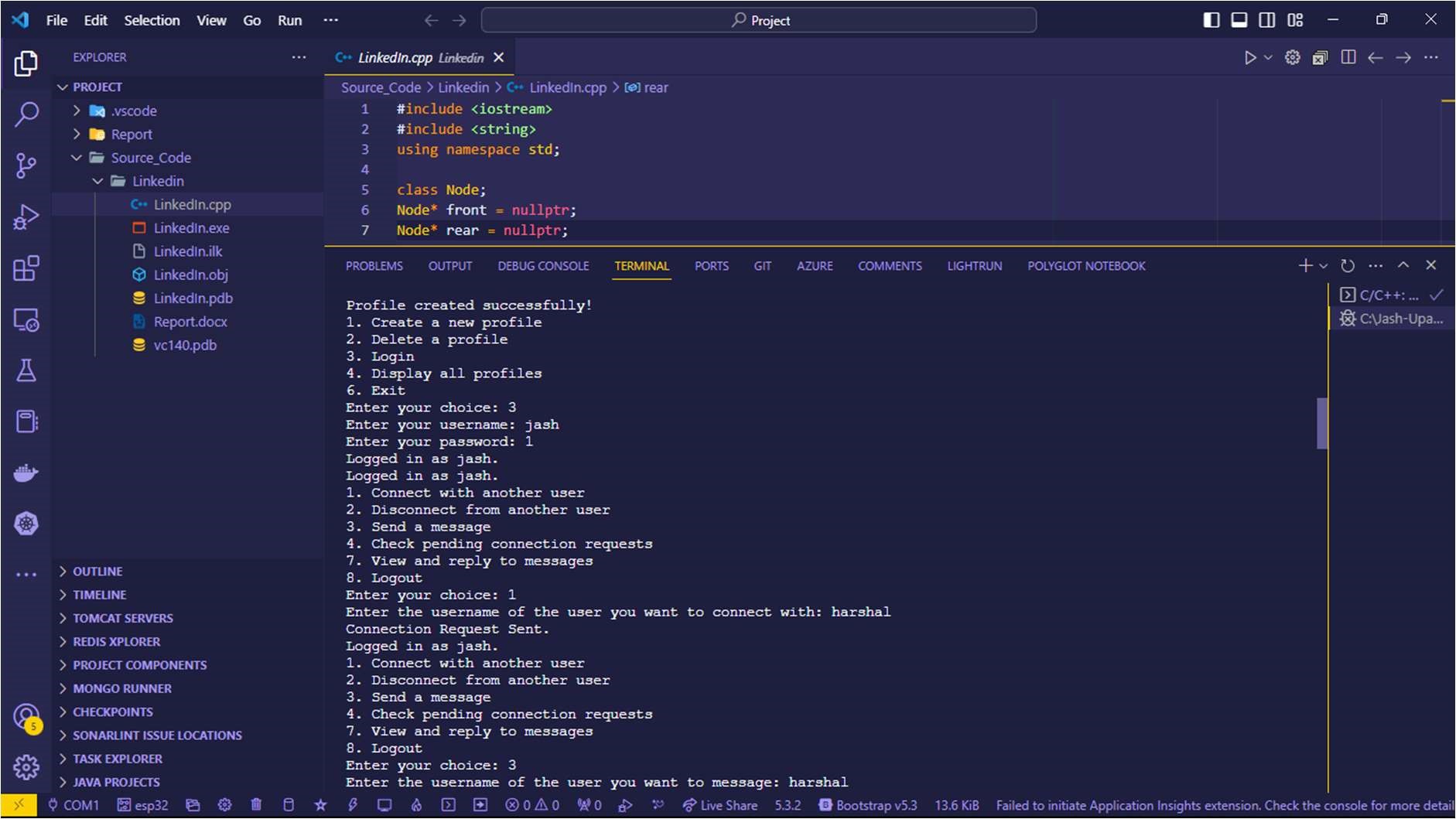
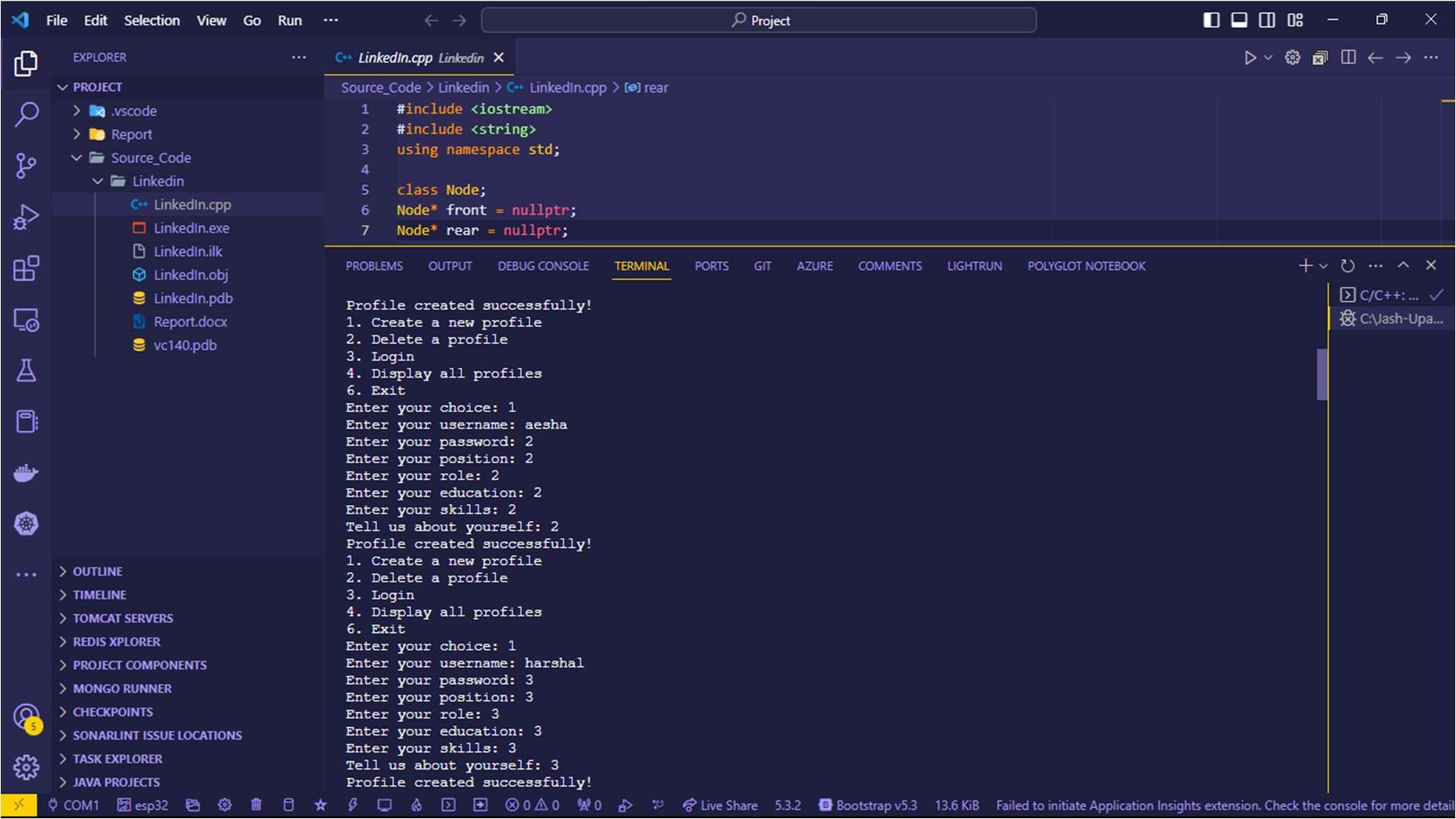
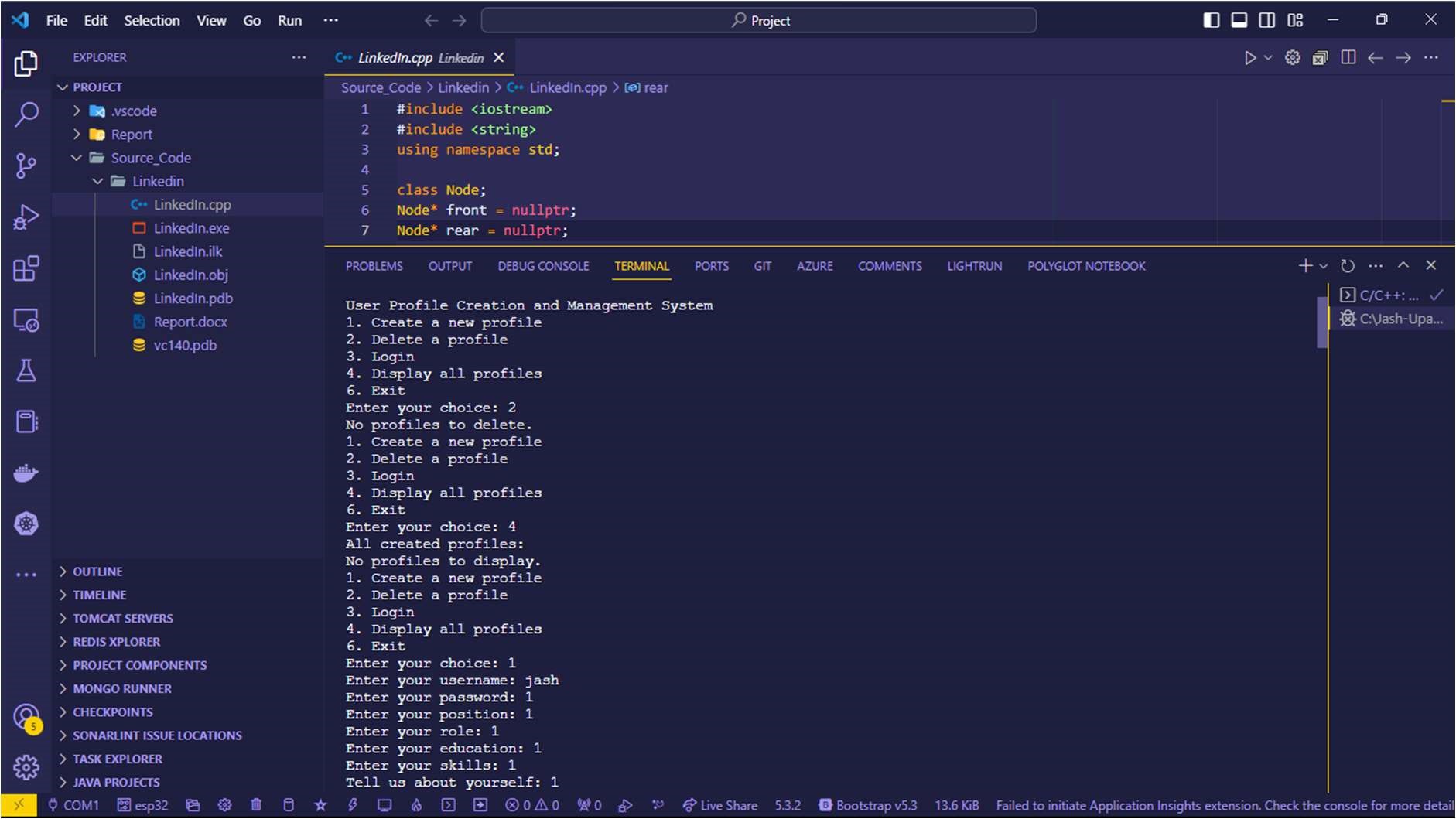
} while (choice != '6');

return 0; }

4.2 Screenshots 4.2.1. Code:



4.2.2. Input And Output:



# Chapter 5: Learning, Conclusion and Future Scope 5.1 Your learning

Our Learnings were:

1. Implementation of data structures.
2. effective work distribution among a team
3. how to handle compilers for complex projects
4. professional coding standards
5. Debugging

# 5.2 Your experience

Experience was unique and value adding as it made us more passionate towards programming. The happiness when the code runs after solving an error is unparalleled and also learnt a lot about the programming eco system such as compilers/data structures etc

# 5.3 Conclusion

In summary, the "Mock LinkedIn Implementation" project is a versatile platform that empowers users to manage profiles, connect, communicate, and explore job opportunities. Leveraging various data structures, it offers practical functionalities for networking and professional interactions. This project has real-world applications in online networking and job-seeking, showcasing the importance of data structures in building functional social networking applications

# 5.4 Future scope

Future scope is [functionalities which can be added are]:

a list of additional functionalities that can be added to enhance the "Mock LinkedIn Implementation" project:

* Recommendation System:

Implement a recommendation engine to suggest potential connections, job postings, and relevant content to users.

* Job Search and Filters:

Allow users to search for jobs and apply filters based on location, industry, and job type.

* User Interactions Feed:

Create a feed where users can see updates and posts from their connections.

* Skill Endorsements:

Enable users to endorse each other's skills and provide recommendations.

* Privacy Settings:

Implement privacy controls for user profiles and content, allowing users to manage who can view their information.

* Event Management:

Add features for users to create, manage, and promote professional events or webinars.