# SE RELEASE REPORT

TEAM 10

## I. Introduction :-

### A. BACKGROUND :

Software development relies heavily on the use of various dependencies, including libraries, frameworks, and modules. These dependencies allow developers to create complex applications and systems more efficiently. However, managing dependencies can be challenging, especially when dealing with large and complex software projects.The failure to manage dependencies can lead to system failures and security vulnerabilities.

Therefore, Our **Team 10** has worked on Constructing a Tool named **"Dependalytics"** for comprehending Dependencies present in any zipped folder(no need to extract the files by the user) .

### B. PURPOSE OF THE TOOL :

The purpose of this tool is to help developers and software teams manage dependencies more effectively. Main benefit of this tool is —> **it provides a comprehensive view of dependencies within a software system**, allowing developers to identify potential issues and proactively manage dependencies.

### C. SCOPE OF THE TOOL :

For Now, The tool is designed to work on **Detecting C++ libraries/Dependencies**. It can be used by individual developers or software teams based on their Requirement.Our Team is planning on extending the Detection of multiple language Dependencies.

# II. Dependencies :-

## A. DEFINITION OF DEPENDENCIES :

**A software component cannot function properly in isolation and needs other software components to work together in order to operate correctly. Such interrelated software components are referred to as dependencies and can take various forms such as libraries, frameworks, modules, or other components that are essential for the system's proper functioning.**

## B. TYPES OF DEPENDENCIES :

There are several types of dependencies, including runtime dependencies, development dependencies, and test dependencies. Runtime dependencies are required for a system to function properly, while development and test dependencies are only required during the development and testing phases of a project.

**Dependencies in software development can be categorized into several types, including:**

1. **Direct dependencies:** These are dependencies that a software system explicitly relies on and are included as dependencies in the project's build configuration files.
2. **Transitive dependencies:** These are dependencies of direct dependencies and are not explicitly included in the project's build configuration files. Transitive dependencies are managed by the build system automatically.
3. **Build-time dependencies**: These are dependencies that are required during the build process to compile, test, and package the software system.
4. **Runtime dependencies:** These are dependencies that are required for a software system to function correctly during runtime.

5. **Test dependencies:** These are dependencies that are only required during the testing phase of a software project and are not needed for the system to function in production.

Next we have the **Types of C++ libraries**, i.e; In C++, there are different types of libraries used for various purposes. Here are some of the most common types:

- **Standard libraries :** These are the libraries that are included in the C++ programming language and provide basic functionality, such as input/output, string manipulation, math functions, etc.

- **Third-party libraries :** These are libraries developed by third-party developers and not included in the standard C++ library. They provide additional functionality and can be downloaded and installed separately.

- **Template libraries :** These libraries provide generic data structures and algorithms that can be used with different data types. They are usually implemented using templates.

- **Object-oriented libraries :** These libraries are designed using object-oriented principles and provide classes and objects that encapsulate functionality and data.

- **Graphics libraries :** These libraries provide functions and classes for creating graphics and animations, such as OpenGL and DirectX.

- **Networking libraries :** These libraries provide functionality for networking, such as sending and receiving data over the internet, such as Boost.Asio.

- **Database libraries :** These provide functions and classes for interacting with databases, such as MySQL and SQLite.

### C. IMPORTANCE OF MANAGING DEPENDENCIES :

Managing dependencies plays a crucial role in software development for several reasons :

1. Firstly, it helps to maintain system stability and security.
2. Secondly, it facilitates developers in keeping up with the latest software components and security patches.
3. Lastly, it empowers developers to proactively manage dependencies and recognize potential issues.

### D. CHALLENGES OF MANAGING DEPENDENCIES :

Managing dependencies can be challenging because dependencies can be complex and interdependent. Additionally, frequent changes to dependencies can further complicate the task of staying up to date.

## III. Tool Features :-

### A. OVERVIEW OF THE TOOL :

This tool designed by our Team provides the user a comprehensive view of dependencies within a software system/Project. This Tool was mainly built to Detect the inbuilt C++ Dependencies present in a system.For now, Our target was to capture all the dependencies - External & internal.

It allows developers to visualize dependencies and track changes over time. It also states all the Dependencies present in the Project based on External & internal Factors. It also shows the **Header Files,External Dependencies & Dependencies based on various linkings.**

★ **GITHUB Link :** https://github.com/GroshanCS10/SE_Tool-1.git

**Header Files :** A header file in C++ is a file that contains declarations of various program elements such as functions, classes, and variables, which can be utilized in a C++ program. Typically, header files are identified with a ".h" extension. The primary purpose of header files is to enable C++ programs to utilize code from external libraries or modules without requiring knowledge of their implementation details. Instead, header files define the interface of the library or module by specifying the names and signatures of its functions and classes. As a result, C++ programs can utilize the code by simply including the corresponding header file in their source code.

**Target-Link Dependencies :** In C++, target-link dependencies can be specified in different ways. Two commonly used methods are using header files to declare the interface of a component and its dependencies, or using build tools like make or CMake to automate the build process and specify the dependencies between components. Other techniques, such as preprocessor macros and command line arguments, can also be used to specify dependencies, but using header files and build tools is generally considered a best practice.

## B. FUNCTIONALITY :

The tool provides several key functionalities, including:

- ➔ **Dependency analysis:** The tool analyzes the dependencies within a software system and identifies potential issues.
- ➔ **Dependency visualization:** The tool provides a visual representation of the dependencies within a software system, making it easier to identify potential issues.
- ➔ **Dependency management:** The tool enables developers to manage dependencies more effectively by tracking changes and proactively identifying potential issues.

☐ **We need to have at least one cpp file in the project you're about to upload, otherwise The Tool won't accept the Zipped folder and shows error.**

- ➔ We will be storing the extracted folder locally in the BackEnd so that We can parse the extracted C++ files to find the Dependencies and Display them.
- ➔ All the Dependencies are being captured in the .Json file present in the BackEnd.
- ➔ Also, We will then display those Libraries/Dependencies present in Json file on the Website, So that the users can clearly get the Idea of Dependencies they are about to use/using.
- ➔ Finally, the user will be able to view all sorts of Dependencies/Libraries present in their project.

★ **All the below noted files/dependencies will be checked by our Tool to ensure there are no missing Libraries.**

```
if not file.endswith(".cpp") and not file.endswith(".cc") and
not file.endswith(".cxx") and not file.endswith(".h") and not
file.endswith(".hpp") and not file.endswith(".hh") and not
file.endswith(".hxx") and not file.endswith(".cu") and not
file.endswith(".cuh") and file != "CMakeLists.txt":
```

## C. USER INTERFACE :

The user interface is intuitive and easy to use. Developers can quickly navigate through the tool and access the information they need. It is very user-friendly and is suggested by our Team to use the tool as many times as possible to get a feel of it.

We have used FIGMA to specialize our way of displaying Dependencies/Libraries without any hassle.

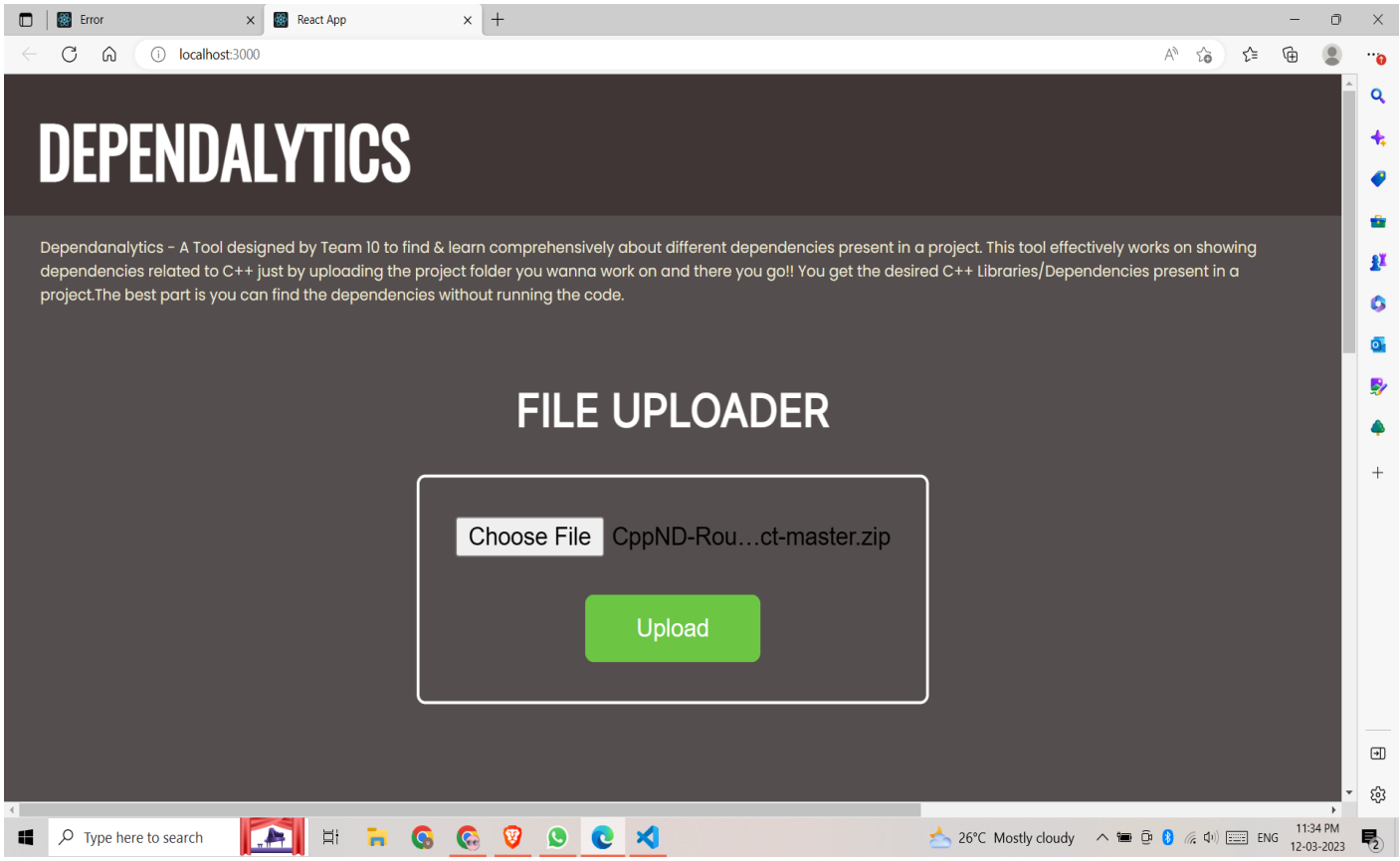
Figma Link :
https://www.figma.com/file/l4UteEwTXgfqVKmH73Su2k/Untitled?node-id=1%3A2&t=Gpi8VwTUAZWUe7bF-1

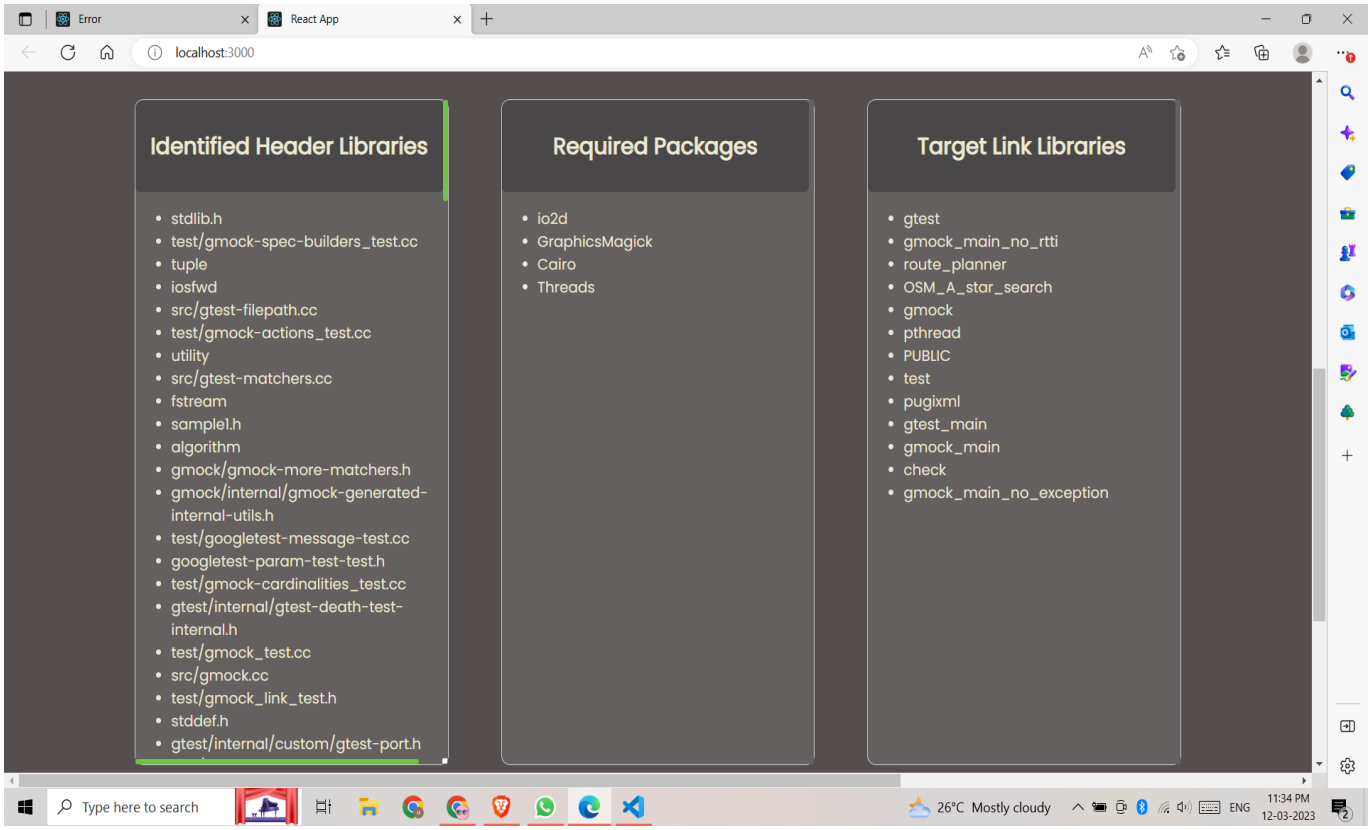
## D. SYSTEM REQUIREMENTS :

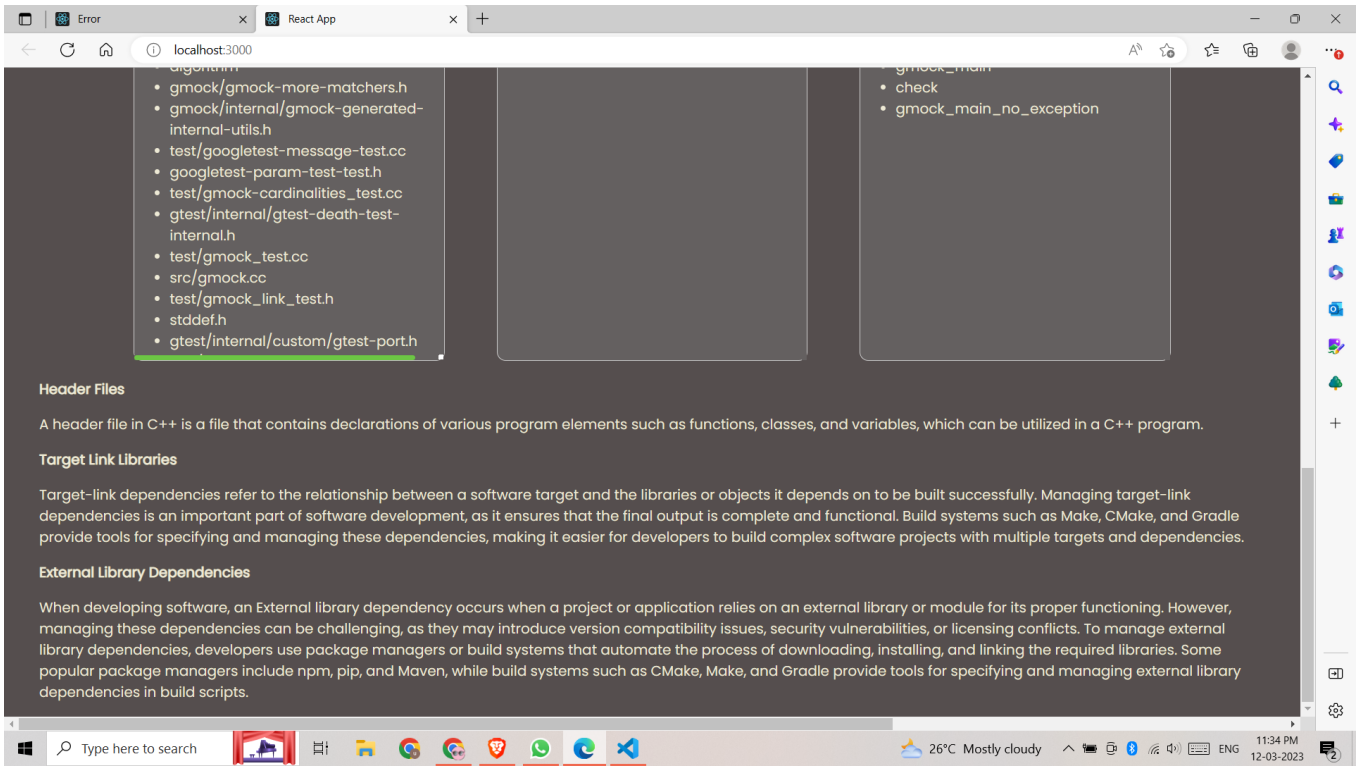
The tool is basically built in a Website made with React and therefore, it works on Any system as long as users are comfortable with uploading the Zipped folder into our Tool.

This Website is Simple and made Specially to Read and Detect the Dependencies immediately after users upload the Zipped Folder/Project into our Tool.

# HOW TOOL WORKS :




Error
React App
localhost:3000
Identified Header Libraries
stdlib.h
test/gmock-spec-builders_test.cc
tuple
iosfwd
src/gtest-filepath.cc
test/gmock-actions_test.cc
utility
src/gtest-matchers.cc
fstream
sample1.h
algorithm
gmock/gmock-more-matchers.h
gmock/internal/gmock-generated-internal-utils.h
test/googletest-message-test.cc
googletest-param-test-test.h
test/gmock-cardinalities_test.cc
gtest/internal/gtest-death-test-internal.h
test/gmock_test.cc
src/gmock.cc
test/gmock_link_test.h
stddef.h
gtest/internal/custom/gtest-port.h
Required Packages
io2d
GraphicsMagick
Cairo
Threads
Target Link Libraries
gtest
gmock_main_no_rtti
route_planner
OSM_A_star_search
gmock
pthread
PUBLIC
test
pugixml
gtest_main
gmock_main
check
gmock_main_no_exception
Type here to search
26°C Mostly cloudy
ENG
11:34 PM
12-03-2023


Error
React App
localhost:3000
gmock/gmock-more-matchers.h
gmock/internal/gmock-generated-internal-utils.h
test/googletest-message-test.cc
googletest-param-test-test.h
test/gmock-cardinalities_test.cc
gtest/internal/gtest-death-test-internal.h
test/gmock_test.cc
src/gmock.cc
test/gmock_link_test.h
stddef.h
gtest/internal/custom/gtest-port.h
check
gmock_main_no_exception
Header Files
A header file in C++ is a file that contains declarations of various program elements such as functions, classes, and variables, which can be utilized in a C++ program.
Target Link Libraries
Target-link dependencies refer to the relationship between a software target and the libraries or objects it depends on to be built successfully. Managing target-link dependencies is an important part of software development, as it ensures that the final output is complete and functional. Build systems such as Make, CMake, and Gradle provide tools for specifying and managing these dependencies, making it easier for developers to build complex software projects with multiple targets and dependencies.
External Library Dependencies
When developing software, an External library dependency occurs when a project or application relies on an external library or module for its proper functioning. However, managing these dependencies can be challenging, as they may introduce version compatibility issues, security vulnerabilities, or licensing conflicts. To manage external library dependencies, developers use package managers or build systems that automate the process of downloading, installing, and linking the required libraries. Some popular package managers include npm, pip, and Maven, while build systems such as CMake, Make, and Gradle provide tools for specifying and managing external library dependencies in build scripts.
Type here to search
26°C Mostly cloudy
ENG
11:34 PM
12-03-2023

### E. LANGUAGES USED :

- ☐ **React Js**
- ☐ **Javascript**
- ☐ **Node Js**
- ☐ **HTML**
- ☐ **CSS**
- ☐ **Express Js**
- ☐ **Python**

## IV. Case Studies :-

### A. EXAMPLES OF HOW THE TOOL HAS HELPED MANAGE DEPENDENCIES :

The tool has helped teams manage dependencies more effectively by providing a comprehensive view of dependencies within a software system. It has enabled teams to identify potential issues and proactively manage dependencies, leading to more stable and secure software systems.

➔ One case Study/Example where we found it useful :

★ To capture external library dependencies
★ Check for the build system of the project
★ It can be done by detecting CmakeLists.txt [CMake] or configure.ac/configure.in [Autotools] or Make/Make.am [Make].

- ○
- ○ CMakeLists.txt:
- ○
- ○ Find the *find_package* - Needs to be installed
- ○ *link_directories* or *add_libraries* or *target_link_libraries* - needs to be installed and linked
- ○
- ○ For version cmake_minimum_required(VERSION 3.10)
- ○
- ○ Example:
- ○ For the given project

- https://github.com/theWrongCode-dev/CppND-Route-Planning-Project
- 
- find_package(io2d REQUIRED)
- find_package(Cairo)
- find_package(GraphicsMagick)
- target_link_libraries(test gtest_main route_planner pugixml)

## FUTURE WORK :

- We are working on **Making it available to work on multiple Language projects**.
- We will be working on directly **Accepting the GITHUB links** or either the zipped folder downloaded from other sources/GITHUB.
- We will try **inserting the Documentation links of those Libraries** obtained using the hover option.
- We will be **showing multiple versions of the libraries/Dependencies** obtained and **producing links to the latest version of the Dependency/Library** that is being used.
- We will be **generating a TXT file** which can be downloaded in the user Local system and therefore , the Userwill be able to see clearly all the types & versions of Dependencies/Libraries present in their project.