Final Project Presentation

Each project group will give a 15-20-minute presentation of their final project. You should prepare a 15-minute talk and assume that you will get 10 minutes of questions. Your presentation will be similar in style and organization to your project presentation. Each team member should talk about his/her presentation part.

Obviously if you give your presentation on September 3rd, you may have some parts of your project that are not done. If this is the case, you should include a discussion of the unfinished parts of your project and what you need to do to complete them. Try to at least get some experimental results from the completed parts of your project that you can present in your talk.

Tips on Topics That You Should Include in Your Final Year Project

1. Do not miss writing or mentioning the name of each team member.
2. Do not read the above details in the presentation.
3. Present your Problem definition and company details which include as Name of the company, since how many years it exists, branches if the company has, how many employees working in the company, details of the area on which company works.
4. You should cover Problems in the current system, the solution to an existing problem that you offer, hardware and software requirements and version details that you have used, what exclusive features you have added in your project.
5. If the complexity of your project is higher, you need to add the system flow chart to explain the flow of your project.
6. Include diagrams: UML Use case diagram, Class diagram, Sequence diagram, and so on.
7. Use global use case diagram that explains all the modules and functionality of all the modules. If the diagram is complicated, explain each module individually by adding a link of each module diagram. If required, use it.

Checklist for Good Programming

Category	**Checklist Item**
	Identifiers
Make sure all your identifiers are meaningful	
One-letter identifiers are almost never meaningful.	
Names like flag and temp are seldom meaningful. Instead of flag, consider naming the Boolean condition it checks for, such as valueFound.	
Use Solution/Problem Domain Meaningful Names	
Consider multi-word identifiers, like nameIndex. Long identifiers (within reason) tend to be very readable.	
	Modularization
A program is built out of interacting components	
Don't put all your code into the main() routine.	
In fact, do not make any routine do too much work. If it is longer than about 50 lines, it is maybe too long.	

If you duplicate code several times, consider whether a loop would work better, or perhaps a subroutine.	
	Formatting
Your program should be easy to read.	
Do follow a consistent indentation pattern that reflects the program's control structure.	
	Coding
You want your coding to be clear, maintainable, and efficient, in that order. Some of the rules here are very specific; others are more general.	
Don't use a sequence of if statements that have no else if only one can match; use else if.	
In a switch statement, always check for the default case. Likewise, in a sequence of if-then-else statements, use a final else.	
Use loops to initialize data structures if possible.	
Use each variable and each field of a structure for exactly one purpose. Do not overload them unless there is an excellent reason to do so.	
Don't use the same identifier for both a type, a variable, and a file name, even if you change the capitalization. It's too confusing.	
Try not to use global or nonlocal variables. Declare each variable in the smallest scope you can.	
Do not Repeat Yourself (Avoid Duplication)	
	Documentation
It helps you as you write the program.	
Add documentation as you write the program. You can always modify it as your design changes.	
Include internal documentation: What algorithms and data structures are you using?	
Check your whole program and documentation for spelling mistakes.	
	Exceptions
Use checked exceptions for recoverable conditions and runtime exceptions for programming errors	
Favor the use of standard exceptions	
Do not ignore exceptions	
Check parameters for validity	Methods
Methods should be small!	
Return empty arrays or collections, not nulls	
Minimize the accessibility of classes and members	Classes and Interfaces
Classes should be small!	
In public classes, use accessor methods, not public fields	
Minimize the scope of local variables	General Programming
Refer to objects by their interfaces	General Programming

References:

- Java Programming Style Guidelines https://petroware.no/html/javastyle.html