DAY 2 SUMMARY

3. Technical Orientation & Internship Project
3.1 Revision of Software Development Life Cycle (SDLC)

The session began with a revision of the Software Development Life Cycle (SDLC), highlighting how software is built through a series of structured phases. These phases include requirement gathering, system design, development, testing, deployment, and ongoing maintenance. This structured flow ensures that software is developed systematically, risks are minimized, and quality is maintained throughout the project lifecycle.

3.2 Introduction to CI/CD and Automation

Modern development practices such as Continuous Integration and Continuous Delivery/Deployment (CI/CD) were discussed. Continuous Integration refers to the practice of frequently merging code changes into a shared repository, allowing teams to detect issues early. Continuous Delivery and Deployment focus on automating the build, testing, and release processes so that software can be delivered reliably and quickly.

Jenkins was introduced as an automation server that plays a key role in CI/CD pipelines. It helps by pulling code from version control systems, building the application, running automated tests, and supporting the process of moving changes toward deployment. This reduces manual effort and increases consistency in software releases.

3.3 Understanding Deployment Environments

Different deployment environments used in real-world software development were also explained. The Test environment is used by QA teams to validate functionality and identify defects. The Staging environment closely resembles the production setup and is used for final verification before release. The Production environment is the live system where end users interact with the application.

User Acceptance Testing (UAT) was described as a crucial phase in which business users verify that the system meets real-world requirements and expectations before it goes live.

3.4 GitHub Branching Strategy in Team Development

The session also covered GitHub branching strategies as a way to support parallel development. Developers create separate branches for new features or bug fixes, allowing multiple team members to work simultaneously without affecting the main codebase. These changes are later reviewed and merged into the main branch through pull requests, helping maintain code stability, organization, and collaboration efficiency.