Sl.No.	Module Name
1	Computer Architecture - Hardware, Network and Software
2	Software Development Life Cycle and Agile Principles
3	Programming Constructs and Algorithms for Problem Solv
4	Linux Operating System and Source Code Management w
5	RDBMS and SQL
6	Core Java Programming
7	HTML, CSS and Javascript
8	Web and J2EE
9	Android Framework with Java
10	Kotlin Programming
11	Android Framework with Kotlin
12	iOS with Swift, SwiftUI Development
13	Appium Testing
14	Capstone Project
TOTAL	

Duration (Days)		
1		
3		
0		
2		
0		
4		
2		
0		
12		
5		
10		
12		
7		
5		
63		

Module
Duration (Days
Sl. No.
1.0
1.1
1.2
1.3
1.4
1.5
2.0
2.1
2.2
2.3
2.4
2.5

Computer Architecture - Hardware, Network and Software
1
Topics
Understanding Computer Architecture
Introduction to Computer Architecture
Overview of computer architecture.
Importance in software development.
Basic terminology.
Central Processing Unit (CPU)
CPU components and functions.
How a CPU executes instructions.
CPU performance factors.
Memory Hierarchy
Types of memory (RAM, Cache, Hard Drives).
How data is stored and accessed.
Memory management concepts.
Input/Output Systems
Overview of I/O systems.
Communication between CPU, memory, and I/O devices.
Introduction to buses and data transfer.
Basic Concepts in Parallelism and Hardware Acceleration
Introduction to parallel computing.
Multi-core processors and GPUs.
Real-world applications and examples.
Understanding Computer Networking
Introduction to Networking
Basic networking concepts.
Importance in software development and communication.
Network Models and Protocols
OSI and TCP/IP models.
Common protocols: HTTP, FTP, TCP, UDP.
IP Addressing and Subnets
IPv4 vs. IPv6.
Subnetting basics.
Network Address Translation (NAT).
Wireless and Wired Networks
Comparison of wired and wireless networking.
WiFi, Ethernet, and emerging technologies.
Network Security Basics
Introduction to network security challenges.
Basic security measures (Firewalls, VPNs).

Back

Day End Assignment Options

Assignment 1: Draw your Home Network Topology and explain how you are accessing the RPS Lab environment.

Assignment 2: Identify a real-world application for both parallel computing and networked systems. Explain how these technologies are used and why they are important in that context.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.
3. Provide references for any external sources used.

Submit your work in a single PDF document by end of Module

Remarks
Understanding the System from a Software Developer perspective, the learners must understand different components of a modern computing device (Computers, Mobiles, Embedded Systems, Intelligent Systems etc.). How the differents parts of the system function, how the data is stored and retrieved. How parallel computing works, different computing solutions for solving complex problems with parallel computing. How data is moved in paralled systems.
Understanding the Networks from a Software Developer perspective, the learners must understand how computer network works, how data is transferred across two connected systems in a network, different addressing schemes. What are protocols, commonly used protocols and their use. How does network isolation and topology work, how data traverses over internet, what is NAT and why they must know it. Understand network security, firewall, VPN etc.

uration (Days
0.5
0.5

Module
Duration (Days
Sl. No.
1.0
1.2
2.0
2.1
2.2
2.3
2.4
3.0
3.1
3.2

3.3
4.0
4.1
4.2
4.3
5.0
5.1
5.2
5.3
6.0
6.1
6.2
7.0
7.1

7.2

Software Development Life Cycle and Agile Principles
3.50

Topics
Introduction to Software Development Life Cycle (SDLC)
Definition and Importance of SDLC
Explaining what SDLC is and its role in software development.
The significance of SDLC in managing and controlling software development projects.
Key Phases of SDLC
Requirement Gathering: Identifying project goals, requirements, and constraints.
Design: Planning the architecture and interface of the software based on requirements.
Implementation (Coding): Actual coding of the software according to the design.
Testing: Verifying the software against requirements to ensure it is bug-free.
Deployment: Releasing the final product to the user or market.
Maintenance: Updating and maintaining the software post-deployment.
Exploring SDLC Models
Waterfall Model
Sequential, phase-dependent model.
Each phase must be completed before the next begins.
Best suited for projects with well-defined requirements.
Agile Methodologies
Emphasizes iterative development and flexibility.
Encourages customer involvement and feedback.
Suitable for projects with changing requirements.
Spiral Model
Combines iterative development with systematic, risk-driven approaches.
Focuses on early identification and mitigation of risks.
Ideal for large, complex projects with significant risks.
V-Model
Known as the Verification and Validation model.
Each development phase has a corresponding testing phase.
Emphasizes the importance of testing in every phase of development.
Modern Development Methodologies
Test-Driven Development (TDD)
Develop tests for each small functionality before writing the code.
Ensures the software is built with testing in mind, leading to fewer bugs.
Encourages simple designs and inspires confidence in software reliability.
Behavior-Driven Development (BDD)
Focuses on the behavior of an application for the end-user.
Facilitates communication between developers, QA teams, and non-technical stakeholders.
Helps ensure the development process aligns with business goals.
Feature-Driven Development (FDD)

Emphasizes delivering tangible, working software repeatedly in a timely manner.
Focuses on building and designing features.
Encourages status reporting at all levels, ensuring clarity and accountability.
Agile Principles and Communication
Agile Introduction and Values
Introduction to Agile values
Individuals and interactions, Working solutions
Customer collaboration, Responding to change
Agile principles (e.g., prioritizing customer needs, iterative development).
Scrum Framework
Overview of Scrum framework
Roles (Product Owner, Scrum Master, Development Team) and responsibilities.
Scrum artifacts (Product Backlog, Sprint Backlog, Increment).
Scrum ceremonies - Sprint Planning, Daily Standup, Sprint Review, Sprint Retrospective
Kanban and Lean
Understanding Kanban and Lean principles.
Visualizing work with Kanban boards.
Set up a Kanban board for a real-world scenario.
Reducing waste in software development (Muda, Mura, Muri).
Agile Methodologies and Effective Communication
Agile Planning and Estimation
Agile planning techniques (backlog grooming, story points).
Plan and estimate a project using Agile methods.
Creating and prioritizing user stories.
Daily Standup and Communication
Conducting effective Daily Standup meetings.
Common challenges and solutions in Daily Standup communication.
Sprint Review and Stakeholder Communication
The importance of Sprint Reviews in Agile.
Preparing for and conducting a Sprint Review.
Effective Communication in SDLC
Design and Development Communication
Collaborative communication during the design and development phases.
Cross-functional teams and their role in communication.
Testing and Reporting
Effective communication during testing phases.
Test-driven development (TDD) and its impact on communication.
Testing, Deployment, Retrospectives, and Conclusion
Integration and System Testing
Communication strategies during integration and system testing.
Test environments and data management.
Reporting and tracking defects.
Deployment and Release Communication

	Effective communication strategies during deployment.
	Deployment planning, rollback strategies, and contingency plans.
	Post-deployment monitoring and communication.

Back

Day End Assignment Options

Assignment 1: SDLC Overview - Create a one-page infographic that outlines the SDLC phases (Requirements, Design, Implementation, Testing, Deployment), highlighting the importance of each phase and how they interconnect.

Assignment 2: Develop a case study analyzing the implementation of SDLC phases in a real-world engineering project. Evaluate how Requirement Gathering, Design, Implementation, Testing, Deployment, and Maintenance contribute to project outcomes.

Assignment 3: Research and compare SDLC models suitable for engineering projects. Present findings on Waterfall, Agile, Spiral, and V-Model approaches, emphasizing their advantages, disadvantages, and applicability in different engineering contexts.

Assignment 1: Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.

Assignment 2: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

No Assignment
Assignment 1: Agile Project Planning - Create a one-page project plan for a new software feature using Agile planning techniques. Include backlog items with estimated story points and a prioritized list of user stories. **Assignment 2:** Daily Standup Simulation - Write a script for a Daily Standup meeting for a development team working on the software feature from Assignment 1. Address a common challenge and incorporate a solution into the communication flow.
No Assignment

Remarks
In this session, learners will understand the fundamental concepts of Software Development Life Cycle (SDLC), including its definition, importance, and role in managing software projects. They will delve into key phases such as Requirement Gathering, Design, Implementation, Testing, Deployment, and Maintenance, gaining insight into each stage's significance and activities. Furthermore, learners will explore various SDLC models like Waterfall, Agile, Spiral, and V-Model, comprehending their unique approaches and suitability for different project scenarios. Understanding SDLC equips learners with essential project management skills, ensuring efficient development, quality assurance, and successful project delivery, making it indispensable knowledge for aspiring software developers.
In this session, learners explore modern development methodologies: Test-Driven Development (TDD), Behavior-Driven Development (BDD), and Feature-Driven Development (FDD). TDD prioritizes tests before code, ensuring reliability and simplicity. BDD focuses on end-user behavior, aiding collaboration and alignment with business goals. FDD emphasizes iterative delivery of functional software, promoting feature building and clear reporting for accountability. These methodologies enhance software quality, communication, and alignment with business goals, vital for modern development practices.

Learners will explore Agile methodologies, focusing on its foundational values and principles. They will gain insights into Agile's core values, which prioritize individuals and interactions, working solutions, customer collaboration, and responding to change. Understanding these Agile principles is vital, as they introduce a flexible and customer-centric approach to software development, enabling teams to adapt to evolving requirements and deliver value efficiently. This knowledge equips learners with a modern, adaptable framework for software development, ensuring they can work effectively in dynamic environments and provide value to customers and stakeholders.
Learners will dive into Agile Development methodology. They will learn how Agile teams plan and prioritize work, with a focus on backlog grooming and story point estimation. This knowledge is essential for learners, as it equips them with the skills needed to contribute to Agile teams effectively. They will be able to participate in Agile meetings, understand how work is organized, and understand the effort required for different tasks. This understanding ensures that they can collaborate seamlessly with their teams and contribute to the successful delivery of Agile projects.
Learners will explore effective communication within Agile teams. They will learn to participate in Daily Standup meetings, Sprint Reviews, and stakeholder communication. This knowledge is crucial, as it enables them to engage in transparent and collaborative communication, ensuring that the team stays aligned, and project progress is effectively conveyed to stakeholders.
Learners will delve into the communication strategies required during software testing and deployment. They will learn how to effectively report defects and coordinate deployment activities. This knowledge is essential for learners as it ensures smooth testing, defect resolution, and successful software deployment, reducing post-release issues.

Duration (Days)
1.00
0.75

0.50
0.50
0.25
0.50

Module
Duration (Days
Sl. No.
1.0
1.1
1.2
1.3
1.4
1.5
2.0

2.1
2.2
2.3
2.4
2.5
3.0
3.1

3.2
3.3
3.4

Linux Operating System and Source Code Management with Git
2.5

Topics
Linux Basics and Command Line
Introduction to Linux
Linux History and Philosophy
Origins of Linux, GNU/Linux
Open Source Movement
Popular Linux Distributions
Overview of Ubuntu, Fedora, CentOS
Understanding Desktop Environments
GNOME, KDE, XFCE
Command Line Basics
Terminal Fundamentals
Accessing the Terminal
Basic Navigation Commands
Directory Structures
Home, Root, Bin, and Other Directories
Absolute vs Relative Paths
File Management Commands
Creating, Listing, Deleting Files and Directories
Working with Files and Text Editors
File Operations
Copying, Moving, Removing Files
Searching Within Files with grep
Introduction to Nano and Vim
Basic Operations: Opening, Editing, Saving Files
Vim Modes and Navigation
Basic File System Structure and Permissions
Exploring Linux File System Hierarchy
Standard Directories and Their Roles
File Permissions and Ownership
Reading and Modifying File Permissions
Changing File Ownership
Remote System Access
Accessing remote system using SSH and Telnet access
Copy Files from/to remote System
Source Code Management with Git
Version Control with Git
Git Introduction
Why Version Control?

Git Installation and Configuration
Git Basics
Initializing a Repository (git init)
Staging and Committing Changes (git add, git commit)
Git Operations and Workflow
Managing Changes
Viewing Commit History (git log)
Undoing Changes (git revert, git reset)
Branching in Git
Creating and Switching Branches (git branch, git checkout)
Basic Merging Techniques
Branching and Merging
Advanced Branching Strategies
Feature Branching, Hotfix Branches
Complex Merging Techniques
Resolving Complex Merge Conflicts
Rebase vs Merge
Collaborative Development with Git
Collaborative Workflows in Git
Forking and Cloning Repositories
Managing Collaborative Changes
Working with Branches Remotely
Pushing and Pulling Branches
Remote Branch Management
Code Review and Collaboration Practices
Utilizing Pull Requests
Creating and Managing Pull Requests
Code Review Process
Shell Scripting with Bash
Bash Scripting Basics
Shell Scripting Basics
Creating and Running Shell Scripts
Script File Structure
Making Scripts Executable
Basic Shell Script Examples
Variables and Data Types
Environment Variables
Custom Variables
Data Types and Declaration
Working with Strings and Numbers
Control Structures and Functions
Control Structures
Conditional Statements (if, then, else, elif)

Loop Structures (for, while, until)
Case Statements
Functions in Bash
Defining and Calling Functions
Function Parameters and Return Values
Scope of Variables in Functions
Advanced Scripting Techniques
Advanced Bash Features
Arrays and Associative Arrays
Reading and Writing Files
Redirecting Input and Output
Using Pipes and Filters
Error Handling
Exit Status of Commands
Trap Statements for Error Detection
Debugging Bash Scripts
Text Processing Tools
Essential Text Processing Commands
grep, sed, awk
cut, sort, uniq, tr
Regular Expressions Basics
Using Text Processing Tools in Scripts
Integrating grep, sed, awk in Shell Scripts
Practical Examples of Text Processing

Back
Day End Assignment Options
No Assignment
Assignment 1: Initialize a new Git repository in a directory of your choice. Add a simple text file to the repository and make the first commit. **Assignment 2:** Branch Creation and Switching Create a new branch named 'feature' and switch to it. Make changes in the

'feature' branch and commit them.

Assignment 3: Feature Branches and Hotfixes
Create a 'hotfix' branch to fix an issue in the main code. Merge the 'hotfix' branch into 'main' ensuring that the issue is resolved.

Assignment 1: Ensure the script checks if a specific file (e.g., myfile.txt) exists in the current directory. If it exists, print "File exists", otherwise print "File not found".

Assignment 2: Write a script that reads numbers from the user until they enter '0'. The script should also print whether each number is odd or even.

Assignment 3: Create a function that takes a filename as an argument and prints the number of lines in the file. Call this function from your script with different filenames.

Assignment 4: Write a script that creates a directory named TestDir and inside it, creates ten files named File1.txt, File2.txt, ... File10.txt. Each file should contain its filename as its content (e.g., File1.txt contains "File1.txt").

Assignment 5: Modify the script to handle errors, such as the directory already existing or lacking permissions to create files.
Add a debugging mode that prints additional information when enabled.

Assignment 6: Given a sample log file, write a script using grep to extract all lines containing "ERROR". Use awk to print the date, time, and error message of each extracted line.
Data Processing with sed

Assignment 7: Create a script that takes a text file and replaces all occurrences of "old_text" with "new_text". Use sed to perform this operation and output the result to a new file.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.
3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.

Remarks
Participants will learn about Linux's background, how to use the command line, manage files, and edit texts using tools like Nano and Vim. They'll also learn how to search inside files with grep and access a remote Linux System. Knowing these basics of Linux is important for learners, as it helps them work better and faster in software development and managing computer systems. These skills are the first step to becoming good at more advanced tech work.
In this session on Source Code Management with Git, learners will understand why version control is crucial and how Git facilitates it. They will set up Git, learn to initialize repositories, and manage code changes. With hands-on practice, they'll master viewing commit history and reversing changes. The course will also cover branching, including creation, switching, and merging

strategies. Advanced techniques such as managing complex merges and resolving conflicts will be taught. Learners will engage in collaborative Git workflows, including forking, cloning, and remote branch management. Finally, they'll hone skills in code review and collaboration through pull requests, essential for team-based projects. This knowledge is vital for maintaining code quality and efficient team collaboration.
In this session, you'll learn about Bash Shell Scripting and Text Processing in Linux, which are essential skills in the computer world today. Starting with basic Linux commands, you'll learn how to write scripts, manage files, and process data. This session is very practical for anyone using computers, especially for tasks like automating routine work, analyzing data, and managing systems. By the end of the session, you'll be able to handle complex tasks more easily and efficiently, saving time and effort. This training is especially valuable for those in IT, data analysis, and system administration, making their work smoother and more efficient.

Duration (Days)
0.75

0.75

1

Module
Duration (Days
Sl. No.
1.0
1.1
1.2
2.0
2.1
2.2
3.0
3.1
4.0
4.1
4.2
4.3
4.4
5.0
5.1
6.0

6.1
6.2
6.3
6.4
7.0
7.1
7.2
7.3
7.4
7.5
8.0
8.1
8.2
8.3
9.0
9.1
9.2
9.3
9.4
9.5

10.0
10.1
10.2
11.0
11.1
11.2
12.0
12.1
12.2
12.3
13.0
13.1
13.2
13.3
14.0

14.1
14.2
14.3
15.0
15.1
15.2
15.3
15.4
15.5

15.6
15.7

Core Java Programming
10
Topics
Java Fundamentals
Java Environment Setup
Installing JDK
Configuring IDEs (Eclipse, IntelliJ IDEA)
Java Syntax and Basic Constructs
Primitive Data Types and Variables
Operators and their Precedence
Control Flow Statements (if, switch, loops - for, while, do-while)
Java Object-Oriented Programming (OOP) Basics
Object-Oriented Programming (OOP)
Classes and Objects: Definitions and Differences
Constructors: Purpose and Types
Encapsulation: Access Modifiers and Getters/Setters
Inheritance: Superclass and Subclass Relationships
Polymorphism: Method Overloading and Overriding
Organizing Code
Packages: Creating and Using Packages
Classpath: Understanding and Setting the Classpath
Java Modules: Modular Programming in Java
Exception Handling
Exception Handling
try-catch Blocks: Basic Exception Handling
Creating Custom Exceptions: When and How to Create
Java Collections Framework and Generics
Arrays: Declaration, Initialization, and Usage
Introduction to List interface.
Exploring ArrayList and LinkedList.
Understanding the Set interface.
Working with HashSet, LinkedHashSet, and TreeSet.
Differentiating between various Set implementations.
Introduction to Map interface.
Exploring HashMap, LinkedHashMap, and TreeMap.
Key-value pair management and use cases.
Iterators and Comparators
String Handling
String Manipulation: Methods in the String Class
StringBuilder and StringBuffer: Mutability and Performance
Arrays in Java

Definition and characteristics
Implement Basic operations
Access
Insert
Delete
Update
Reverse
Slice
Sorting an Array - Brute Force Method
Searching an Array - Linear Search Method
Concurrency and Multithreading in Java
Multithreading Basics: Creating and Managing Threads
Thread Lifecycle: States and Transitions
Synchronization and Inter-thread Communication
Thread Synchronization: Synchronized Blocks and Methods
Thread Pools and Concurrency Utilities
Concurrency Utilities: Executors, Concurrent Collections, CompletableFuture
Thread Safety: Writing Thread-Safe Code, Immutable Objects
Advanced Java Features
Generics and Type Safety
Generic Classes and Methods
Bounded Type Parameters and Wildcards
Generic Collections and Type Inference
Java Reflection and Annotations
Reflection API: Accessing and Modifying Runtime Classes
Annotations: Built-in and Custom Annotations
Lambda Expressions and Functional Programming
Functional Programming Concepts
Lambda Expressions: Syntax and Usage
Functional Interfaces: Predicate, Function, Consumer, Supplier
Method References
Java IO, NIO, and Networking
Java IO Basics
File Handling (FileReader, FileWriter)
BufferedReader, BufferedWriter
Advanced Java IO
Streams and File I/O
Serialization and Deserialization
New IO (NIO)
Channels, Buffers, Selectors
File and Directory Operations
Java Networking
Sockets, URL, InetAddress

Building Client-Server Applications
Java 8 Date and Time API
Working with LocalDate, LocalTime, LocalDateTime
Temporal Adjusters, Periods, and Time Zones
JDBC Programming
Basics of JDBC
JDBC overview and architecture
Establishing database connections
Executing SQL queries using JDBC
Processing query results with ResultSet
Advanced JDBC
Using PreparedStatement for efficient queries
Handling database metadata
Executing stored procedures with CallableStatement
Transaction management in JDBC
Java Testing and Debugging
JUnit Framework
Basics of JUnit for unit testing
Annotations: @Test, @Before, @After, @BeforeClass, @AfterClass
Writing test cases and assertions: assertEquals, assertTrue, assertFalse
Organizing tests into suites
Debugging Techniques
Using IDE debugging features (Eclipse, IntelliJ)
Setting and using breakpoints
Inspecting variables and expressions
Analyzing call stacks
Utilizing logging frameworks (Log4j, SLF4J)
Code Profiling and Performance Analysis
Tools for profiling: VisualVM, JProfiler
Memory usage and leak detection
CPU usage and performance bottlenecks
Analyzing garbage collection and JVM behavior
Memory Management
JVM Memory Structure
Heap, Stack, Method Area, and their roles
Object creation and lifecycle
Memory Leak Identification and Handling
Common causes and symptoms of memory leaks
Tools for analysis (Eclipse MAT, VisualVM)
Garbage Collection
Overview of GC algorithms (Serial, Parallel, CMS, G1, ZGC)
Tuning and optimizing garbage collection
Java Design Patterns

Creational Patterns
Singleton
Use cases in Java
Factory Method
Scenarios for usage in Java
Builder
Application in complex object construction
Structural Patterns
Adapter
Proxy
Usage in controlling object access
Behavioral Patterns
Observer
Event handling and listener models in Java
Strategy
Use in altering object behavior
Command
Application in encapsulating invocation
Java Build System with Maven
Introduction to Maven
Overview of Build Tools
What is Maven and Why Use It?
Core Concepts: POM, Coordinates, Repositories, Plugins
Installation and Setup
Downloading and Installing Maven
Configuring Maven on Different Operating Systems
Setting up Environment Variables
Verifying Maven Installation
Maven Features and Project Structuring
Understanding the Standard Directory Layout
Exploring the POM File
Overview of Maven Goals and Phases
Essential Maven Commands
Archetype Generation
Understanding Maven Archetypes
Creating a New Project Using an Archetype
Custom Archetypes
Maven Build Lifecycle
Deep Dive into the Build Lifecycle
The Clean, Default, and Site Lifecycles
Commonly Used Lifecycle Phases
Dependency Management
Understanding Dependency Management in Maven

  - Declaring Dependencies in the POM File
  - Transitive Dependencies and Conflict Resolution
  - Managing Plugin Dependencies
- Integrating Maven with Eclipse
  - Installing Maven Integration for Eclipse (m2e)
  - Importing Maven Projects into Eclipse
  - Running Maven Builds from Eclipse
  - Debugging and Troubleshooting Maven in Eclipse

Back
Remarks
Learner will understand the essentials of setting up a Java development environment and the syntax required to write Java programs. They'll learn to configure tools like JDK and IDEs, which are vital for code development and execution. The engineer will also delve into the core principles of object-oriented programming, understanding classes, objects, and inheritance, which are fundamental in crafting structured and maintainable code. Additionally, they will acquire skills in managing code execution flow and error handling through control statements and exception handling, crucial for building robust applications.
Learner will learn to efficiently manage data in Java using the Collections Framework, mastering arrays, lists, sets, maps, and their respective interfaces and implementations. They will also understand generics for type-safe operations. Proficiency in iterators and comparators for collection traversal and sorting is covered. Additionally, skills in string manipulation through various String class methods and the performance benefits of StringBuilder and StringBuffer are emphasized.
Building on their understanding of functions in Java, learners will apply this

foundational knowledge to arrays, understanding their structure and manipulation. They will work with array operations—accessing, inserting, deleting, updating, and reversing elements—to grasp basic data handling. Learners will also explore sorting algorithms, starting with brute force methods, and searching techniques, such as the linear search, to find elements efficiently. This combination of array manipulation and algorithmic thinking is critical for practical applications in software development, enabling learners to manage and process data effectively in real-world programming scenarios.
Learner will acquire skills to create, manage, and synchronize threads, crucial for building efficient, high-performance applications. They'll understand thread lifecycles, coordination, and advanced utilities like executors, ensuring robust thread safety. These competencies are vital for developing complex, responsive software that fully utilizes system capabilities.
Learners will explore generics for type-safe code and learn to use generic classes, methods, and collections to create flexible and reusable code components. They'll delve into Java's reflection API to interact with and modify runtime classes, and annotations to provide metadata within code. They will be introduced to lambda expressions and functional programming concepts, enabling more concise and expressive code. Understanding functional interfaces and method references will further streamline their code, promoting a functional approach to solving problems and manipulating data. These advanced skills are essential for writing clean, maintainable, and efficient Java code.
Learning Java IO and NIO is essential for handling files and directories, enabling learners to read, write, and manage data on a storage medium. Advanced IO covers streams and file operations, while NIO introduces channels and buffers for more efficient data handling. Networking teaches the creation of networked applications using sockets and protocols. The Java Date and Time API provides a comprehensive framework for managing dates, times, and time zones, crucial for applications that require time-sensitive operations. These capabilities are foundational for developing a wide range of Java applications, from desktop to server-side programming.

Learner will gain an understanding of JDBC programming, They'll explore JDBC to connect Java applications to a database, execute queries, and handle results. Advanced JDBC techniques such as using PreparedStatements and CallableStatements improve query efficiency and enable transaction management, while handling database metadata allows for dynamic database interaction, all critical skills for backend development in Java.
Java Testing and Debugging are crucial skills for Learners to ensure the quality and reliability of their Java applications. Learning JUnit provides a framework for unit testing, enabling the creation of test cases with annotations like @Test and assertion methods like assertEquals. Debugging techniques involve using IDE features like breakpoints, variable inspection, and call stack analysis to identify and fix issues efficiently. Engineers also explore logging frameworks like Log4j and performance analysis tools like VisualVM and JProfiler to optimize code for memory usage, CPU performance, and garbage collection efficiency, contributing to the creation of robust and high-performing Java applications.
Memory management is vital for Java performance. Learners delve into JVM memory structure, enabling efficient object handling. They master memory leak detection using tools like Eclipse MAT and VisualVM, spotting issues promptly. Understanding garbage collection algorithms like Serial, Parallel, CMS, G1, and ZGC equips them to optimize collection processes, enhancing memory usage and application responsiveness. These skills are crucial for developing efficient and reliable Java applications.
Java design patterns are essential for learners to create efficient and

maintainable code. They will explore creational patterns like Singleton, Factory Method, and Builder to manage object creation effectively. Structural patterns such as Adapter and Proxy control object access and improve code flexibility. Behavioral patterns like Observer, Strategy, and Command enable the alteration of object behavior and encapsulation of invocations. Engineers will apply these patterns to solve common design problems, making their Java applications more organized and adaptable.
Learners will understand Maven's automation capabilities for Java projects, enabling efficient builds and dependency management. They'll learn to structure projects, utilize archetypes for boilerplate code, and master the build lifecycle to enhance project predictability. Understanding Maven is vital for Java developers to maintain consistent build processes, save time on project setup, and easily collaborate across teams. Mastery of Maven also aids in integrating with IDEs like Eclipse, streamlining development workflows. This curriculum is designed to empower developers with the tools and practices needed to leverage Maven effectively in real-world scenarios.

Duration (Days)		
2		
0.75		

1.00		
0.75		
0.75		
1		

1		
0.75		
0.5		

1		
0.5		

Day End Assignment Options

Day 1 and 2:
Task 1: Data Types/Variables
Write a program that declares two integer variables, swaps their values without using a third variable, a
the result.
Task 2: Operators
Create a program that simulates a simple calculator using command-line arguments to perform and pr
result of addition, subtraction, multiplication, and division..
Task 3: Control Flow
Write a Java program that reads an integer and prints whether it is a prime number using a for loop an
statements.
Task 4: Constructors
Implement a Matrix class that has a constructor which initializes the dimensions of a matrix and a meth
the matrix with values.
Task 5: Inheritance
Create a Shape class with a method area() and extend it with Circle and Rectangle classes overriding
method appropriately.
Task 6: Packages/Classpath
Create a package com.math.operations and include classes for various arithmetic operations. Demons
to compile and run these using the classpath.
Task 7: Basic Exception Handling
Write a program that attempts to divide by zero, catches the ArithmeticException, and provides a custo
message.

Day 3:
Task 1: Arrays - Declaration, Initialization, and Usage
Create a program that declares an array of integers, initializes it with consecutive numbers, and prints
in reverse order.
Task 2: List interface
Implement a method that takes a List as an argument and removes every second element from the lis
prints the resulting list.
Task 3: Set interface
Write a program that reads words from a String variable into a Set and prints out the number of unique
demonstrating the unique property of sets.
Task 4: Map interface
Create a Java class that uses a Map to store the frequency of each word that appears in a given string

Task 5: Iterators and Comparators
Write a custom Comparator to sort a list of Employee objects by their salary and then by name if the sa
same.

Day 4:
Task 1: Array Sorting and Searching

a) Implement a function called BruteForceSort that sorts an array using the brute force approach. Use function to sort an array created with InitializeArray.
b) Write a function named PerformLinearSearch that searches for a specific element in an array and r index of the element if found or -1 if not found.

Task 2: Two-Sum Problem
a) Given an array of integers, write a program that finds if there are two numbers that add up to a spec You may assume that each input would have exactly one solution, and you may not use the same elem Optimize the solution for time complexity.

Task 3: Understanding Functions through Arrays
a) Write a recursive function named SumArray that calculates and returns the sum of elements in an a demonstarte with example.

Task 4: Advanced Array Operations
a) Implement a method SliceArray that takes an array, a starting index, and an end index, then returns array containing the elements from the start to the end index.
b) Create a recursive function to find the nth element of a Fibonacci sequence and store the first n elem an array.

Day 5:
Task 1: Creating and Managing Threads
Write a program that starts two threads, where each thread prints numbers from 1 to 10 with a 1-secon between each number

Task 2: States and Transitions
Create a Java class that simulates a thread going through different lifecycle states: NEW, RUNNABLE WAITING, TIMED_WAITING, BLOCKED, and TERMINATED. Use methods like sleep(), wait(), notify() to demonstrate these states..

Task 3: Synchronization and Inter-thread Communication
Implement a producer-consumer problem using wait() and notify() methods to handle the correct proce sequence between threads.

Task 4: Synchronized Blocks and Methods
Write a program that simulates a bank account being accessed by multiple threads to perform deposits withdrawals using synchronized methods to prevent race conditions.

Task 5: Thread Pools and Concurrency Utilities
Create a fixed-size thread pool and submit multiple tasks that perform complex calculations or I/O ope observe the execution.

Task 6: Executors, Concurrent Collections, CompletableFuture
Use an ExecutorService to parallelize a task that calculates prime numbers up to a given number and CompletableFuture to write the results to a file asynchronously.

Task 7: Writing Thread-Safe Code, Immutable Objects
Design a thread-safe Counter class with increment and decrement methods. Then demonstrate its usa

multiple threads. Also, implement and use an immutable class to share data between threads.

Day 6:
Task 1: Generics and Type Safety
Create a generic **Pair** class that holds two objects of different types, and write a method to return a rev
version of the pair.

Task 2: Generic Classes and Methods
Implement a generic method that swaps the positions of two elements in an array, regardless of their t
demonstrate its usage with different object types.

Task 3: Reflection API
Use reflection to inspect a class's methods, fields, and constructors, and modify the access level of a p
field, setting its value during runtime

Task 4: Lambda Expressions
Implement a Comparator for a Person class using a lambda expression, and sort a list of Person objec
age.

Task 5: Functional Interfaces
Create a method that accepts functions as parameters using Predicate, Function, Consumer, and Sup
interfaces to operate on a Person object.

Day 7 and 8:
Task 1: Java IO Basics
Write a program that reads a text file and counts the frequency of each word using FileReader and File

Task 2: Serialization and Deserialization
Serialize a custom object to a file and then deserialize it back to recover the object state.

Task 3: New IO (NIO)
Use NIO Channels and Buffers to read content from a file and write to another file.

Task 4: Java Networking

Task 3: PreparedStatement
Modify the SELECT query program to use PreparedStatement to parameterize the query and prevent
injection.

Day 8:
Task 1: Write a set of JUnit tests for a given class with simple mathematical operations (add, subtract,
divide) using the basic @Test annotation.

Task 2: Extend the above JUnit tests to use @Before, @After, @BeforeClass, and @AfterClass anno
manage test setup and teardown.

Module	
Duration (Days	

Sl. No.
1.0
1.1
1.2
1.3
2.0
2.1
2.2
2.3
3.0
3.1
3.2
4.0
4.1

4.2
4.3
5.0
5.1
5.2
5.3
5.4
6.0
6.1
6.2
6.3
6.4

HTML, CSS and Javascript
2.5
Topics
Understanding HTML5 and CSS3
HTML5 Basics
Understanding the structure of an HTML document
Working with HTML tags and attributes
Exploring layout tags and semantic tags
Working with tables, forms, and frames
Utilizing style and div tags for layouts
CSS3 Basics
Understanding CSS selectors, properties, and values
Implementing CSS box model
Working with CSS layouts: flexbox and grid
HTML5 and CSS3 Advanced Features
Implementing HTML5 semantic elements
Working with CSS3 transitions and animations
Understanding responsive design with CSS3 media queries
Working with Bootstrap for responsive design
Introduction to Bootstrap
Understanding the Bootstrap framework
Setting up a project with Bootstrap
Bootstrap Grid System
Understanding the grid system
Implementing responsive layouts with the grid system
Bootstrap Components
Working with Bootstrap components like Navbar, Carousel, Modal, etc.
Customizing Bootstrap components
Writing JavaScript and understanding ES6 features
JavaScript Basics
Understanding JavaScript syntax and data types
Implementing control flow with conditional statements and loops
Defining and calling functions
JavaScript ES6 Features
Working with let and const
Understanding arrow functions
Implementing classes in JavaScript
Working with modules
Understanding the Document Object Model (DOM)
Introduction to the DOM
What is the DOM?

Understanding the DOM tree
Manipulating the DOM
Selecting elements with querySelector and querySelectorAll
Modifying element content with textContent and innerHTML
Changing element attributes and styles
DOM Events
Understanding event-driven programming
Adding event listeners to DOM elements
Implementing event handlers
Implementing asynchronous JavaScript: Callbacks, Promises, Async/Await
Understanding Asynchronous JavaScript
Synchronous vs asynchronous programming
Understanding the event loop
Working with Callbacks
What is a callback function?
Implementing asynchronous operations with callbacks
Understanding callback hell
Promises
What is a Promise?
Creating and consuming Promises
Chaining Promises with then and catch
Async/Await
Understanding async functions
Using the await keyword
Handling errors with try/catch
Working with AJAX
Introduction to AJAX
What is AJAX?
How does AJAX work?
Benefits of using AJAX
Making AJAX Requests
Creating an XMLHttpRequest object
Configuring the request with open
Sending the request with send
Handling AJAX Responses
Understanding the readyState and status properties
Handling the onreadystatechange event
Parsing JSON responses
Error Handling in AJAX
Handling network errors
Handling HTTP errors

Back

Day End Assignment Options
Assignment 1: Implement a navigation bar using an unordered list with links to different sections of your HTML page. Use CSS to style the list as a horizontal menu and highlight the current page or section. **Assignment 2:** Create a simple HTML page that includes the use of headings, paragraphs, and at least two semantic tags like <article> or <section>. Add a table with data of your choice and a form with fields for a user's name, email, and a submit button. **Assignment 3:** Apply the CSS box model to ensure that your page content has appropriate margins and padding. Create a layout using div tags and style them to arrange content in a multi-column format using floats or flexbox. **Assignment 4:** Enhance the page by adding CSS3 animations to the menu and form elements. Also, use media queries to make the page responsive, ensuring it looks good on both desktop and mobile screen sizes. **Assignment 5:** Take the static HTML page from Day 1 and integrate Bootstrap. Refactor the navigation bar, table, and form to use Bootstrap components. Ensure the page is responsive using Bootstrap's grid system.
Assignment 1: Create a JavaScript object representing a user with properties for name, email, and age. Write functions that manipulate this object, such as changing the name, updating the email, and calculating the user's birth year. **Assignment 2:** Use regular expressions in JavaScript to validate the email address entered in the form. It should check for the correct format of the email and display a message to the user if the format is incorrect.

Remarks
In this section, participants will learn the essentials of web development, starting with the basics of HTML5 and CSS3. They'll understand the structure of an HTML document, work with HTML tags and attributes, and implement HTML forms and input types. They'll also learn about CSS selectors, properties, values, the CSS box model, and layouts using flexbox and grid. The section covers advanced features of HTML5 and CSS3, including semantic elements, transitions, animations, and responsive design with media queries. Participants will also work with Bootstrap for responsive design, understanding the framework, setting up a project, and working with the grid system and components. These skills are fundamental for modern web development.
In this section, participants will learn the basics of JavaScript and its ES6 features, including syntax, data types, control flow, functions, let and const, arrow functions, classes, and modules. They'll understand the Document Object Model (DOM), how to manipulate it, and handle DOM events. The section also covers asynchronous JavaScript, including callbacks, promises, async/await, and AJAX. Participants will learn how to make AJAX requests, handle responses, and manage errors. These skills are fundamental for modern web development, making them a must-learn for any aspiring web developer.

Duration (Days)
1.00

1.50

Module	
Duration (Days	

Sl. No.
1.0
2.0
2.1
2.2
2.3
3.0
3.1

3.2
4.0
4.1
4.2
4.3
4.4
5.0
5.1
5.2

5.3
6.0
6.1
6.2
6.3
6.4
6.5

Web and J2EE
10.00

Topics
Servlet/JSP
Introduction to Servlets
Understanding the role of servlets in web applications
Servlet lifecycle and HTTP request handling
Introduction to JSP
Transitioning from servlets to JSP
JSP syntax and directives
Integrating JavaBeans in JSP for better data management
JSP syntax and directives
Integrating JavaBeans in JSP for better data management
Spring Core
Spring Framework Introduction
Comprehensive overview of Spring Framework's capabilities
Discussion on the benefits of using Spring in enterprise applications
Detailed examination of the core features of Spring
Exploration of Spring Framework's layered architecture
Bean Lifecycle and Factory
In-depth analysis of the Bean Factory
Detailed workings of the Application Context
Lifecycle management of Spring Beans
Advanced Dependency Injection techniques and Inversion of Control principles
Dependency Injection (DI) Types
Detailed comparison of Setter-based DI vs. Constructor-based DI
Strategic Autowiring of Beans including by Type, Name, and Constructor
Utilization and best practices of Annotations for Autowiring
Spring MVC
Spring MVC Framework
Deep dive into the Spring MVC module and its ecosystem
Role and configuration of the Front Controller in Spring MVC
Comprehensive overview of Spring MVC's components
Dispatcher Servlet and Web Flow
Configuration and roles of the Dispatcher Servlet
Detailed explanation of Handler Mapping beans and strategies
View Resolver mechanisms and its bean configurations
Step-by-step walkthrough of the flow of Spring MVC
Development of Controllers and utilization of Validators
RESTful Services with Spring MVC
Introduction to RESTful Web Services

            - Creating RESTful controllers using **'@RestController'**
            - Mapping web requests with **'@RequestMapping', '@GetMapping' , '@PostMapping', '@PutMapping', '@DeleteMapping'**
            - Understanding HTTP Message Converters
            - Implementing CRUD operations in a RESTful style
            - Exception Handling in RESTful services with **'@ControllerAdvice'** and **'@ExceptionHan**
            - Securing RESTful services with Spring Security
            - Versioning of RESTful services
            - Documentation of RESTful APIs using Swagger or Spring REST Docs
            - Testing RESTful Services
                - Testing strategies for RESTful services
                - Unit and integration testing with Spring Boot Test and @WebMvcTest
                - Mocking web contexts and services with MockMvc
        - Implementation of MVC Forms and data binding

Object Relational Mapping and Hibernate

- ORM and Persistence
    - Detailed definition of Persistence and its role in ORM
    - Identification and resolution of Object-Relational Impedance Mismatch
    - Evaluation of ORM alternatives with a focus on Hibernate
- Hibernate In Practice
    - Hands-on demonstration on JDBC programming for record insertion
    - Step-by-step guide to programming and verifying a Java application with Hibernate
- Hibernate Architecture and CRUD Operations
    - Detailed explanation of Hibernate architecture
    - Learning the Hibernate API including Configuration, SessionFactory, and Session
    - Understanding Object-Relational Mapping in detail
    - Examination of Hibernate Object States
    - Practical guide to making objects persistent, retrieving, modifying, and deleting objects
    - Comprehensive look at CRUD operations in Hibernate
- Querying with Hibernate
    - Techniques for querying with Hibernate Query Language (HQL)
    - Utilizing Criteria API for dynamic query creation
    - Crafting Native SQL queries within Hibernate

Spring Boot and Microservices

- Spring Boot Essentials
    - Exploring the benefits of Spring Boot and its auto-configuration
    - Detailed review of Spring Boot Dependencies and Runtime
    - Utilization of Actuators and DevTools for effective development
- Spring Data JPA and JDBC
    - Overview of JDBC template API and Data Source Configurations
    - Analysis of Maven Dependencies and ORM concepts
    - Entity, ID, and Columns Mapping strategies including Inheritance and Association Mapping
    - RESTful Data Access

Exposing data repositories as RESTful resources with Spring Data REST
Integrating REST with JPA Repositories
Pagination and sorting in RESTful services
HATEOAS and its implementation in Spring Data REST
Microservices with Spring Cloud
Comparative study of Monolithic Architecture vs. Microservices
Deep dive into the 12 factors of cloud-native applications
Service registration and discovery using Eureka and Spring Cloud Bus
Load balancing techniques and declarative REST clients with Feign
Distributed Configuration management with Spring Cloud Config
Reactive Spring
Introduction to Reactive Programming
Defining and differentiating Reactive Programming from Imperative Programming
Discussing the paradigms and importance of Reactive Systems in modern applications
Detailed Reactive Spring Essentials and Streams Specification
Project Reactor and Spring WebFlux
Detailed understanding of Project Reactor's Mono and Flux
Building RESTful APIs using Spring WebFlux
Introduction to reactive RESTful services with WebFlux
Functional endpoints with WebFlux
Backpressure and stream processing in RESTful services
Data Access patterns in Reactive Repositories with Spring Data
R2DBC for Reactive Database Connectivity and transactions management
Testing, Security, and Real-world Use Cases
Strategies for error handling and testing in Reactive Systems
Reactive Security mechanisms including OAuth2 and JWT
Building Reactive Microservices and handling data with WebSockets
Reactive Systems Deployment
Deploying Reactive Applications with tools like Kubernetes
Monitoring and Tracing Reactive Systems for performance and reliability
Advanced Reactive Programming Concepts
Best practices for building Reactive Systems and understanding their limitations
Advanced topics like Server-Sent Events, RSocket, and Project Reactor's debugging features
Future developments in the Reactive landscape and wrapping up the course

Back

Remarks
Learners will learn how Servlets and JSP let them manage websites on the server side. They will understand how Servlets work, handle web requests, and move from Servlets to JSP. They will also learn to use JSP features and JavaBeans to keep data organized, which is key for building good websites.
Learners will understand the Spring Framework, crucial for developing enterprise Java applications. It outlines Spring's functionalities, benefits for enterprise use, and its layered structure. Key topics include Bean lifecycle, factory processes, and application context. The curriculum addresses Dependency Injection, Inversion of Control, different DI methods, and techniques for autowiring Beans. Learners will also understand annotation-based configurations. This knowledge is vital for anyone looking to excel in Java application development using Spring, a standard in the industry for creating scalable, efficient applications.
Learners will engage in an exploring Spring MVC, understanding its comprehensive framework, components, and how they interconnect within the ecosystem. They will configure and utilize the Front Controller, Dispatcher Servlet, and learn to define and use Handler Mappings and View Resolvers. Additionally, they will gain practical experience in developing Controllers, applying Validators, and implementing MVC forms for effective data binding, equipping them with the skills to create dynamic, robust web applications.

Learners will delve into Object-Relational Mapping (ORM) with a focus on Hibernate, where they will define persistence and tackle Object-Relational Impedance Mismatch. They'll gain hands-on experience with JDBC, understand Hibernate’s architecture, and use its API for CRUD operations. The course covers Hibernate's query capabilities, teaching how to use HQL, Criteria API, and native SQL for robust data manipulation, preparing learners to effectively manage and query databases in Java applications.
Learners will explore Spring Boot's streamlined approach to application development with its auto-configuration and runtime management. They'll learn how to leverage Spring Boot's dependencies, Actuators, and DevTools for rapid, efficient development cycles. The curriculum also covers data access with Spring Data JPA and JDBC, including ORM mapping and Maven dependencies. Moving into microservices, learners will compare monolithic architectures with microservices, understand the principles of cloud-native applications, and use Spring Cloud for service discovery, load balancing, and configuration management. This knowledge is vital for building scalable, resilient applications suited for the cloud.

Learners delve into Reactive Programming, distinguishing it from traditional approaches and grasping its critical role in modern applications. They'll explore Reactive Spring's foundations, use Spring WebFlux for creating RESTful APIs, and manage data reactively. The curriculum includes implementing secure, robust microservices, handling real-time data streams, and deploying with Kubernetes. Additionally, it covers advanced reactive patterns, debugging techniques, and prepares participants for emerging trends in the reactive programming domain.

uration (Days)		
1.50		
1.50		

1.50

1.50

2.00

2.00		
2.00		

Project Based Lear

Assignment 1

Development Scenario: Insurance Claim Processing System

Day 1: HTML, CSS, and JavaScript - User Authentication and Profile Setup
Task 1: Design and code the HTML forms for user registration and login, ensuring accessibility standards are met.
Task 2: Apply CSS to style the forms for a consistent look and feel that aligns with the company's branding.
Task 3: Implement JavaScript form validations to provide immediate feedback on user input errors before submission.

Day 2: JavaScript/Bootstrap - Responsive Dashboard for Policy Management
Task 1: Create a dashboard layout with Bootstrap ensuring responsiveness across devices.
Task 2: Utilize Bootstrap's JavaScript components like tabs and modals to enrich the policy management interface.
Task 3: Enhance dashboard interactivity with JavaScript for policy sorting and detailed views.

Day 3: Servlet/JSP, Introduction to JSP - Claims Submission Process
Task 1: Develop Servlets to manage the workflow of submitting insurance claims**.**
Task 2: Construct JSP pages for entering claim information and confirmations.
Task 3: Employ JavaBeans to manage the transition of data in the claim submission process.

Day 4: Spring Core - Policy Administration Backend
Task 1: Refactor policy-related operations to utilize Spring Beans and Dependency Injection.
Task 2: Implement Spring validation on the server side to ensure policy data integrity.
Task 3: Set up Application Context and Bean Factory for a scalable backend structure.

Day 5: Spring MVC - User Claim Interaction Workflow
Task 1: Migrate front-end form handling to Spring MVC controllers.
Task 2: Configure Thymeleaf as the view layer for dynamic content rendering in Spring MVC.
Task 3: Implement data binding and server-side validation within the Spring MVC framework.

Day 6: Object Relational Mapping and Hibernate - Database Integration for Claims and Policies
Task 1: Define Hibernate entity mappings for claim and policy data models.
Task 2: Develop Hibernate DAOs to handle CRUD operations for claims and policies.
Task 3: Write and test HQL and Criteria queries for advanced data retrieval and reporting.

Day 7: Spring Boot and Microservices - Microservices for Claim Processing
Task 1: Transition the monolithic application structure to a microservices architecture using Spring Boot.

using Spring Boot.
Task 2: Implement service discovery with Eureka and develop Feign clients for inter-service communication.
Task 3: Set up and configure Spring Cloud Config for centralized configuration management of microservices.

Day 8: Reactive Spring - Real-time Claim Status Updates
Task 1: Introduce Spring WebFlux for handling real-time claim status updates using reactive streams.
Task 2: Configure R2DBC for reactive database connectivity to update claim status dynamically.
Task 3: Implement WebSocket communication for real-time interaction between the client and the server.

rning Scenario - EP

Assignment 2

Development Scenario: Smart City Transportation Management System

Day 1: HTML, CSS, and JavaScript - User Interface for Route Planning
Task 1: Build the HTML structure for the city's transportation route planner interface.
Task 2: Style the planner interface with CSS for a user-friendly experience across multiple devices.
Task 3: Implement JavaScript to dynamically update route options based on user selections.

Day 2: JavaScript/Bootstrap - Interactive Transit Maps
Task 1: Integrate Bootstrap to develop a responsive layout for interactive transit maps.
Task 2: Use Bootstrap components to display real-time transit data in modals and tooltips.
Task 3: Write JavaScript to handle live updates of transit statuses and to interact with the map.

Day 3: Servlet/JSP, Introduction to JSP - Traffic Data Processing
Task 1: Create Servlets to process real-time traffic data and user queries.
Task 2: Use JSP to present dynamic traffic information and alternative routes.
Task 3: Leverage JavaBeans to store and manage traffic data and user preferences.

Day 4: Spring Core - System Configuration and User Management
Task 1: Configure Spring Beans for user management and session handling.
Task 2: Set up Spring's Dependency Injection to manage services related to traffic data.
Task 3: Establish a secure Application Context for user data processing.

Day 5: Spring MVC - Administration Portal for Transit Management
Task 1: Utilize Spring MVC to create an admin portal for transit officials to manage routes and schedules.
Task 2: Integrate Thymeleaf with Spring MVC for real-time updates and schedule changes.
Task 3: Develop form handling in Spring MVC for incident reporting and user feedback.

Day 6: Object Relational Mapping and Hibernate - Transit Data Modeling
Task 1: Define Hibernate mappings for transit routes, schedules, and vehicle data.
Task 2: Create DAOs using Hibernate for persisting and querying transit operational data.
Task 3: Formulate complex HQL and Criteria API queries for analytics and reporting.

Day 7: Spring Boot and Microservices - Scalable Traffic Monitoring
Task 1: Migrate to Spring Boot for a streamlined setup of microservices for different city zones.
Task 2: Implement Eureka for service discovery among traffic monitoring microservices.
Task 3: Configure Spring Cloud Config for managing microservice settings during peak and off-peak hours.

Day 8: Reactive Spring - Real-Time Alerts and Notifications
Task 1: Apply Spring WebFlux to develop a non-blocking, reactive system for sending

Task 1: Apply Spring WebFlux to develop a non-blocking, reactive system for sending real-time traffic alerts.
Task 2: Use R2DBC for integrating reactive data updates to the traffic management system.
Task 3: Set up WebSocket channels for broadcasting city-wide transportation notifications and updates.

Project Based Learr

Assignment 1

Development Scenario: 5G Network Performance Monitoring System
Project Overview: Develop a microservices-based system to monitor and analyze the performance of a 5G network. This system will collect, process, and visualize data from various network nodes and devices, providing real-time insights into network health, performance bottlenecks, and usage patterns.

Requirement: Network operators need a robust solution to continuously monitor the health and performance of their 5G networks to ensure high quality of service (QoS) for end-users. They require a system that can process vast amounts of data in real time, identify issues proactively, and offer insights into optimizing network performance and capacity planning.
Project Components:
Data Collection Microservice: Captures performance metrics from various 5G network elements (e.g., base stations, network slices) using RESTful APIs. Utilizes the C++ REST SDK for efficient data fetching.
Data Processing Microservice: Processes and aggregates the collected data. Implements algorithms to detect anomalies, performance degradation, and potential bottlenecks using Pistache for the service layer.
Database Microservice: Stores processed data in a structured format for historical analysis and real-time access. Integrates with a NoSQL database (e.g., MongoDB) for scalability and flexibility.
Visualization Microservice: Provides a web-based dashboard for network operators, visualizing key performance indicators (KPIs), historical trends, and real-time data. Uses a C++ library for backend data management and a JavaScript framework (e.g., React) for the frontend.
Notification Microservice: Sends alerts and notifications to network operators based on predefined thresholds or detected anomalies. Integrates with email and SMS gateways for communication.
Deployment & Scaling:
Dockerize each microservice for easy deployment and scaling.
Utilize Docker Compose to manage the multi-container setup.
Implement basic monitoring and logging for each microservice to ensure system reliability and performance.

Day 1: Project Setup and Initial Microservice Development
Task 1: Setup development environment, including Docker, CMake, and Git. Review project requirements and architecture.
Task 2: Begin development of the Data Collection Microservice. Focus on establishing connections to 5G network elements and fetching initial performance metrics.

Day 2: Data Processing and Database Integration

Task 3: Develop the Data Processing Microservice. Implement algorithms for analyzing fetched metrics and identifying anomalies.
Task 4: Implement the Database Microservice. Focus on storing processed data and ensuring efficient data retrieval mechanisms.

Day 3: Visualization and Notification Services
Task 5: Start development of the Visualization Microservice. Create basic dashboard layouts and implement data visualization components.
Task 6: Develop the Notification Microservice. Set up email and SMS integration for alerting based on predefined criteria.

Day 4: Integration and Testing
Task 7: Integrate all developed microservices. Ensure that the system components interact seamlessly, data flows correctly, and the user interface displays data accurately.
Task 8: Conduct initial testing of the entire system. Focus on functional testing, performance testing, and user interface testing.

Day 5: Finalization and Documentation
Task 9: Address any identified issues from testing. Optimize performance based on test results.
Task 10: Document the system architecture, microservices APIs, and user guide. Prepare a demonstration setup.

ing Scenario - NMS

Assignment 2

Devlopment Scenario: Network Configuration and Management System using NETCONF and RESTCONF
Project Overview: Develop a microservices-based network configuration and management system that leverages NETCONF and RESTCONF for managing network devices and services. This system aims to automate the configuration, management, and monitoring of network devices in a large-scale environment, enhancing operational efficiency and reducing manual intervention.

Requirement: Network administrators in large enterprises and service providers are tasked with managing complex networks that include a wide variety of devices and configurations. They need a tool that can simplify the configuration process, automate routine tasks, and provide a clear view of the network's state. The goal is to minimize configuration errors, ensure compliance with industry standards, and quickly adapt to network changes.
Project Components:
Device Configuration Microservice: Interfaces with network devices using NETCONF and RESTCONF protocols to fetch configuration data and apply configuration changes. This microservice will use libraries such as libnetconf2 for NETCONF operations and Pistache for RESTCONF API interactions.
Configuration Database Microservice: Stores current and historical configuration data of network devices. It uses a version-controlled database system to track changes over time and enable rollback if necessary.
Network Monitoring Microservice: Utilizes SNMP and streaming telemetry to monitor the health and performance of network devices. Processes and aggregates monitoring data for real-time visibility.
Compliance and Reporting Microservice: Analyzes device configurations against predefined compliance rules and generates reports on compliance status. Offers insights into non-compliant configurations and potential security vulnerabilities.
Deployment & Scaling:
Containerization of each microservice using Docker to facilitate easy deployment, scaling, and management.
Use of Docker Compose for local development and testing, orchestrating the multi-container application.

Integration Points:
Integration with existing network management tools and databases to ensure seamless data exchange and operational continuity.
Support for multi-vendor network devices by abstracting device-specific details and providing a unified configuration interface.
Key Technologies and Protocols:
NETCONF and RESTCONF for device configuration and management.

SNMP and streaming telemetry for device monitoring.
Docker and Kubernetes for microservices deployment and management.
C++ for backend microservices development, with libraries like libnetconf2 for NETCONF and Pistache for RESTful services.

Day 1: Project Setup and Protocol Integration
Task 1: Set up the development environment. Review project requirements with a focus on NETCONF and RESTCONF protocols.
Task 2: Start development on the Device Configuration Microservice. Implement basic NETCONF and RESTCONF operations to communicate with network devices.

Day 2: Configuration Management and Database Setup
Task 3: Further develop the Device Configuration Microservice to support configuration changes and fetches. Begin integrating with the Configuration Database Microservice.
Task 4: Implement the Configuration Database Microservice. Focus on version control mechanisms and change tracking.

Day 3: Monitoring and Compliance Reporting
Task 5: Develop the Network Monitoring Microservice. Implement data collection via SNMP and telemetry.
Task 6: Start on the Compliance and Reporting Microservice. Develop compliance rules engine and reporting functionalities.

Day 4: User Interface and Alerting Mechanisms
Task 7: Work on the User Interface Microservice. Implement dashboard views for device configurations, compliance reports, and network health.
Task 8: Develop the Alerting Mechanism. Set up alerts for compliance violations and configuration issues.

Day 5: System Integration and Testing
Task 9: Integrate all microservices. Ensure coherent data flow and functionality across the system.
Task 10: Perform comprehensive testing, including integration testing, system testing, and user acceptance testing. Document the system and prepare for a demonstration.

Module
Duration (Days
Sl. No.
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7

1.8
1.9
1.10
2.0
2.1
2.2
2.3

2.4
2.5
2.6
2.7
2.8
3.0
3.1
3.2
3.3
3.4

3.5
3.6
3.7
3.8
4.0
4.1

4.2
5.0
5.1
5.2
5.3
5.4
5.5
6.0
6.1
6.2
6.3
6.4

6.5
6.6
6.7
6.8
6.9
7.0
7.1
7.2
7.3
7.4
7.5

7.6
7.7
7.8
7.9
8.0
8.1

8.1
8.2
8.3
8.4
9.0
9.1
9.2
9.3
9.4

9.5
9.6
9.7
10.0
10.1

10.2
10.3
10.4
10.5
10.6
10.7
10.8

10.9
11.0
11.1
11.2
11.3
12.0
12.1
12.2
12.3
12.4
12.5

12.6
12.7
12.8
13.0
13.1
13.2

13.3
13.5
13.6
13.7
14.0
14.1
14.2
14.3
14.4
14.5

14.6
14.7
14.8
15.0
15.1
15.2
15.3
15.4

15.5
16.0
16.1
16.2
16.3
16.4
17.0
17.1

17.2
17.3
17.4
17.5
17.6
17.7
17.8
18.0
18.1
18.2
18.3
18.4

18.4
18.5

Android Framework with Java
15.00
Topics
Introduction
Android Introduction and Setup
Android Ecosystem and Architecture
Overview of Android OS and its versions
Android architecture components.
Android Studio and Development Tools
Using Android Studio for app development
Creating a new Android project
Navigating the Android Studio interface
Overview of Android Studio features
Debugging, profiling, and emulator.
Android SDK Features
Access to Hardware Including Camera GPS and Sensors
Data Transfers Using Wi-Fi Bluetooth and NFC
Maps Geocoding and Location-Based Services
Background Services
SQLite Database for Data Storage and Retrieval
Shared Data and Inter-Application Communication
Using Widgets and Live Wallpaper to Enhance the Home Screen
Extensive Media Support and 2D/3D Graphics
Cloud to Device Messaging
Optimized Memory and Process Management
Introducing the Open Handset Alliance
What Does Android Run On?
Why Develop for Mobile?
Why Develop for Android?
Factors Driving Android’s Adoption
What Android Has That Other Platforms Don’t Have
The Changing Mobile Development Landscape
Introducing the Development Framework
Understanding the Android Software Stack
The Dalvik Virtual Machine
Android Application Architecture
Android Libraries
Android
Developing for Android
Creating Your First Android Application
Creating a New Android Project

        - Creating an Android Virtual Device
        - Creating Launch Configurations
        - Running and Debugging Your Android Application
        - Understanding Hello World
    - Types of Android Applications
        - Foreground Applications
        - Background Applications
        - Intermittent Applications
        - Widgets and Live Wallpapers
- Developing for Mobile and Embedded Devices
    - Hardware-Imposed Design Considerations
        - Be Efficient
        - Expect Limited Capacity
        - Design for Different Screens
        - Expect Low Speeds High Latency
        - At What Cost?
    - Considering the User’s Environment
    - Being Fast and Efficient
        - Being Responsive
        - Ensuring Data Freshness
        - Developing Secure Applications
        - Ensuring a Seamless User Experience
        - Providing Accessibility
- Android Development Tools
    - The Android Virtual Device Manager
    - Android SDK Manager
    - The Android Emulator
    - The Dalvik Debug Monitor Service
    - The Android Debug Bridge
    - The Hierarchy Viewer and Lint Tool
    - Monkey and Monkey Runner

Creating Applications and Activities

- What Makes an Android Application?
- Introducing the Application Manifest File
    - A Closer Look at the Application Manifest
- Using the Manifest Editor
- Externalizing Resources
    - Creating Resources
        - Simple Values
        - Styles and Themes
        - Drawables
        - Layouts
        - Animations

    - Menus
  - Using Resources
    - Using Resources in Code
    - Referencing Resources Within Resources
    - Using System Resources
    - Referring to Styles in the Current Theme
  - Creating Resources for Different Languages and Hardware
  - Runtime Configuration Changes
- The Android Application Lifecycle
- Understanding an Application’s Priority and Its Process’ States
- Introducing the Android Application Class
  - Extending and Using the Application Class
  - Overriding the Application Lifecycle Events
- A Closer Look at Android Activities
  - Creating Activities
  - The Activity Lifecycle
    - Activity Stacks
    - Activity States
    - Monitoring State Changes
    - Understanding Activity Lifetimes
  - Android Activity Classes

Building User Interfaces

- Fundamental Android UI Design
- Android User Interface Fundamentals
  - Assigning User Interfaces to Activities
- Introducing Layouts
  - Defining Layouts
  - Using Layouts to Create Device Independent User Interfaces
    - Using a Linear Layout
    - Using a Relative Layout
    - Using a Grid Layout
  - Optimizing Layouts
    - Redundant Layout Containers Are Redundant
    - Avoid Using Excessive Views
    - Using Lint to Analyze Your Layouts
- To-Do List Example
- Introducing Fragments
  - Creating New Fragments
  - The Fragment Lifecycle
    - Fragment-Specific Lifecycle Events
    - Fragment States
  - Introducing the Fragment Manager
  - Adding Fragments to Activities

            - Using Fragment Transactions
            - Adding Removing and Replacing Fragments
            - Using the Fragment Manager to Find Fragments
            - Populating Dynamic Activity Layouts with Fragments
            - Fragments and the Back Stack
            - Animating Fragment Transactions
        - Interfacing Between Fragments and Activities
        - Fragments Without User Interfaces
        - Android Fragment Classes
        - Using Fragments for Your To-Do List
    - The Android Widget Toolbox
    - Creating New Views
        - Modifying Existing Views
            - Customizing Your To-Do List
        - Creating Compound Controls
        - Creating Simple Compound Controls Using Layouts
        - Creating Custom Views
            - Creating a New Visual Interface
            - Handling User Interaction Events
            - Supporting Accessibility in Custom Views
            - Creating a Compass View Example
        - Using Custom Controls
    - Introducing Adapters
        - Introducing Some Native Adapters
        - Customizing the Array Adapter
        - Using Adapters to Bind Data to a View
            - Customizing the To-Do List Array Adapter
            - Using the Simple Cursor Adapter

Intents and Broadcast Receivers

- Introducing Intents
    - Using Intents to Launch Activities
        - Explicitly Starting New Activities
        - Implicit Intents and Late Runtime Binding
        - Determining If an Intent Will Resolve
        - Returning Results from Activities
        - Native Android Actions
    - Introducing Linkify
        - Native Linkify Link Types
        - Creating Custom Link Strings
        - Using the Match Filter
        - Using the Transform Filter
    - Using Intents to Broadcast Events
        - Broadcasting Events with Intents

    - Listening for Broadcasts with Broadcast Receivers
    - Broadcasting Ordered Intents
    - Broadcasting Sticky Intents
  - Introducing the Local Broadcast Manager
  - Introducing Pending Intents
- Creating Intent Filters and Broadcast Receivers
  - Using Intent Filters to Service Implicit Intents
    - How Android Resolves Intent Filters
    - Finding and Using Intents Received Within an Activity
    - Passing on Responsibility
    - Selecting a Contact Example
  - Using Intent Filters for Plug-Ins and Extensibility
    - Supplying Anonymous Actions to Applications
    - Discovering New Actions from Third-Party Intent Receivers
    - Incorporating Anonymous Actions as Menu Items
  - Listening for Native Broadcast Intents
  - Monitoring Device State Changes Using Broadcast Intents
    - Listening for Battery Changes
    - Listening for Connectivity Changes
    - Listening for Docking Changes
  - Managing Manifest Receivers at Run Time

Using Internet Resources

- Downloading and Parsing Internet Resources
  - Connecting to an Internet Resource
  - Parsing XML Using the XML Pull Parser
  - Creating an Earthquake Viewer
- Using the Download Manager
  - Downloading Files
  - Customizing Download Manager Notifications
  - Specifying a Download Location
  - Cancelling and Removing Downloads
  - Querying the Download Manager
- Using Internet Services
- Connecting to Google App Engine
- Best Practices for Downloading Data Without Draining the Battery

Files, Saving State, And Preferences

- Saving Simple Application Data
- Creating and Saving Shared Preferences
- Retrieving Shared Preferences
- Creating a Settings Activity for the Earthquake Viewer
- Introducing the Preference Framework and the Preference Activity
  - Defining a Preference Screen Layout in XML
    - Native Preference Controls

        - Using Intents to Import System Preferences into Preference Screens
    - Introducing the Preference Fragment
    - Defining the Preference Fragment Hierarchy
        - Using Preference Headers
    - Introducing the Preference Activity
    - Backward Compatibility and Preference Screens
    - Finding and Using the Shared Preferences Set by Preference Screens
    - Introducing On Shared Preference Change Listeners
- Creating a Standard Preference Activity for the Earthquake Viewer
- Persisting the Application Instance State
    - Saving Activity State Using Shared Preferences
    - Saving and Restoring Activity Instance State
        - Using the Lifecycle Handlers
    - Saving and Restoring Fragment Instance State
        - Using the Lifecycle Handlers
- Including Static Files as Resources
- Working with the File System
    - File-Management Tools
    - Using Application-Specific Folders to Store Files
    - Creating Private Application Files
    - Using the Application File Cache
    - Storing Publicly Readable Files

Databases and Content provviders

- Introducing Android Databases
    - SQLite Databases
    - Content Providers
- Introducing SQLite
- Content Values and Cursors
- Working with SQLite Databases
    - Introducing the SQLiteOpenHelper
    - Opening and Creating Databases Without the SQLiteOpenHelper
    - Android Database Design Considerations
    - Querying a Database
    - Extracting Values from a Cursor
    - Adding, Updating, and Removing Rows
        - Inserting Rows
        - Updating Rows
        - Deleting Rows
- Creating Content Providers
    - Registering Content Providers
    - Publishing Your Content Provider’s URI Address
    - Creating the Content Provider’s Database
    - Implementing Content Provider Queries

    - Content Provider Transactions
    - Storing Files in a Content Provider
    - A Skeleton Content Provider Implementation
- Using Content Providers
    - Introducing the Content Resolver
    - Querying Content Providers
    - Querying for Content Asynchronously Using the Cursor Loader
        - Introducing Loaders
        - Using the Cursor Loader
    - Adding, Deleting, and Updating Content
        - Inserting Content
        - Deleting Content
        - Updating Content
    - Accessing Files Stored in Content Providers
    - Creating a To-Do List Database and Content Provider
- Adding Search to Your Application
    - Making Your Content Provider Searchable
    - Creating a Search Activity for Your Application
    - Making Your Search Activity the Default Search Provider for Your Application
        - Performing a Search and Displaying the Results
    - Using the Search View Widget
    - Supporting Search Suggestions from a Content Provider
    - Surfacing Search Results in the Quick Search Box
- Creating a Searchable Earthquake Content Provider
    - Creating the Content Provider
    - Using the Earthquake Provider
    - Searching the Earthquake Provider
- Native Android Content Providers
    - Using the Media Store Content Provider
    - Using the Contacts Contract Content Provider
        - Introducing the Contacts Contract Content Provider
        - Reading Contact Details
        - Creating and Picking Contacts Using Intents
        - Modifying and Augmenting Contact Details Directly
    - Using the Calendar Content Provider
        - Querying the Calendar
        - Creating and Editing Calendar Entries Using Intents
        - Modifying Calendar Entries Directly

Working in the Background

- Introducing Services
    - Creating and Controlling Services
        - Creating Services
        - Executing a Service and Controlling Its Restart Behavior

    - Starting and Stopping Services
    - Self-Terminating Services
  - Binding Services to Activities
  - Creating Foreground Services
- Using Background Threads
  - Using AsyncTask to Run Asynchronous Tasks
    - Creating New Asynchronous Tasks
    - Running Asynchronous Tasks
  - Introducing the Intent Service
  - Introducing Loaders
  - Manual Thread Creation and GUI Thread Synchronization
- Using Alarms
  - Creating Setting and Canceling Alarms
  - Setting Repeating Alarms
  - Using Repeating Alarms to Schedule Network Refreshes
- Using the Intent Service to Simplify the Application Update Service

Expanding the User Experience

- Introducing the Action Bar
  - Customizing the Action Bar
    - Modifying the Icon and Title Text
    - Customizing the Background
    - Enabling the Split Action Bar Mode
  - Customizing the Action Bar to Control Application Navigation Behavior
    - Configuring Action Bar Icon Navigation Behavior
    - Using Navigation Tabs
    - Using Drop-Down Lists for Navigation
    - Using Custom Navigation Views
  - Introducing Action Bar Actions
- Creating and Using Menus and Action Bar Action Items
  - Introducing the Android Menu System
  - Creating a Menu
  - Specifying Action Bar Actions
  - Menu Item Options
  - Adding Action Views and Action Providers
  - Adding Menu Items from Fragments
  - Defining Menu Hierarchies in XML
  - Updating Menu Items Dynamically
  - Handling Menu Selections
  - Introducing Submenus and Context Menus
    - Creating Submenus
    - Using Context Menus and Popup Menus
- Refreshing an Application Page
- Going Full Screen

Introducing Dialogs
Creating a Dialog
Using the Alert Dialog Class
Using Specialized Input Dialogs
Managing and Displaying Dialogs Using Dialog Fragments
Managing and Displaying Dialogs Using Activity Event Handlers
Using Activities as Dialogs
Let’s Make a Toast
Customizing Toasts
Using Toasts in Worker Threads
Introducing Notifications
Introducing the Notification Manager
Creating Notifications
Creating a Notification and Configuring the Status Bar Display
Using the Default Notification Sounds, Lights, and Vibrations
Making Sounds
Vibrating the Device
Flashing the Lights
Using the Notification Builder
Setting and Customizing the Notification Tray UI
Using the Standard Notification UI
Creating a Custom Notification UI
Customizing the Ticker View
Configuring Ongoing and Insistent Notifications
Triggering, Updating, and Canceling Notifications
Advanced User Experience
Designing for Every Screen Size and Density
Resolution Independence
Using Density-Independent Pixels
Resource Qualifiers for Pixel Density
Supporting and Optimizing for Different Screen Sizes
Creating Scalable Layouts
Optimizing Layouts for Different Screen Types
Specifying Supported Screen Sizes
Creating Scalable Graphics Assets
Color Drawables
Shape Drawables
Gradient Drawables
NinePatch Drawables
Creating Optimized Adaptive and Dynamic Designs
Testing, Testing, Testing
Using Emulator Skins
Testing for Custom Resolutions and Screen Sizes

- Ensuring Accessibility
  - Supporting Navigation Without a Touch Screen
  - Providing a Textual Description of Each View
- Introducing Android Text-to-Speech
- Using Speech Recognition
- Using Speech Recognition for Voice Input
  - Using Speech Recognition for Search
  - Controlling Device Vibration
- Working with Animations
  - Tweened View Animations
    - Creating Tweened View Animations
    - Applying Tweened Animations
    - Using Animation Listeners
    - Animating Layouts and View Groups
  - Creating and Using Frame-by-Frame Animations
  - Interpolated Property Animations
    - Creating Property Animations
    - Creating Property Animation Sets
    - Using Animation Listeners
- Enhancing Your Views
  - Advanced Canvas Drawing
    - What Can You Draw?
    - Getting the Most from Your Paint
    - Improving Paint Quality with Anti-Aliasing
    - Canvas Drawing Best Practice
    - Advanced Compass Face Example
  - Hardware Acceleration
    - Managing Hardware Acceleration Use in Your Applications
    - Checking If Hardware Acceleration Is Enabled
  - Introducing the Surface View
    - When to Use a Surface View
    - Creating Surface Views
    - Creating 3D Views with a Surface View
  - Creating Interactive Controls
    - Using the Touch Screen
    - Using the Device Keys, Buttons, and D-Pad
    - Using the On Key Listener
    - Using the Trackball
- Advanced Drawable Resources
  - Composite Drawables
    - Transformative Drawables
    - Layer Drawables
    - State List Drawables

    - Level List Drawables
- Copy, Paste, and the Clipboard
  - Copying Data to the Clipboard
  - Pasting Clipboard Data

Hardware Sensors

- Using Sensors and the Sensor Manager
  - Supported Android Sensors
  - Introducing Virtual Sensors
  - Finding Sensors
  - Monitoring Sensors
  - Interpreting Sensor Values
- Monitoring a Device’s Movement and Orientation
  - Determining the Natural Orientation of a Device
  - Introducing Accelerometers
  - Detecting Acceleration Changes
  - Creating a Gravitational Force Meter
  - Determining a Device’s Orientation
    - Understanding the Standard Reference Frame
    - Calculating Orientation Using the Accelerometer and Magnetic Field Sensors
    - Remapping the Orientation Reference Frame
    - Determining Orientation Using the Deprecated Orientation Sensor
  - Creating a Compass and Artificial Horizon
  - Introducing the Gyroscope Sensor
- Introducing the Environmental Sensors
  - Using the Barometer Sensor

Maps, Geocoding, and Location-based Services

- Using Location-Based Services
- Using the Emulator with Location-Based Services
  - Updating Locations in Emulator Location Providers
  - Configuring the Emulator to Test Location-Based Services
- Selecting a Location Provider
  - Finding Location Providers
  - Finding Location Providers by Specifying Criteria
  - Determining Location Provider Capabilities
- Finding Your Current Location
  - Location Privacy
  - Finding the Last Known Location
  - Where Am I Example
  - Refreshing the Current Location
  - Tracking Your Location in Where Am I
  - Requesting a Single Location Update
- Best Practice for Location Updates
  - Monitoring Location Provider Status and Availability

- Using Proximity Alerts
- Using the Geocoder
  - Reverse Geocoding
  - Forward Geocoding
  - Geocoding Where Am I
- Creating Map-Based Activities
  - Introducing Map View and Map Activity
  - Getting Your Maps API Key
    - Getting Your Development/Debugging MD5 Fingerprint
    - Getting your Production/Release MD5 Fingerprint
  - Creating a Map-Based Activity
  - Maps and Fragments
  - Configuring and Using Map Views
  - Using the Map Controller
  - Mapping Where Am I
  - Creating and Using Overlays
    - Creating New Overlays
    - Introducing Projections
    - Drawing on the Overlay Canvas
    - Handling Map Tap Events
    - Adding and Removing Overlays
    - Annotating Where Am I
  - Introducing My Location Overlay
  - Introducing Itemized Overlays and Overlay Items
  - Pinning Views to the Map and Map Positions

Invading the Home Screen

- Introducing Home Screen Widgets
- Creating App Widgets
  - Creating the Widget XML Layout Resource
  - Widget Design Guidelines
  - Supported Widget Views and Layouts
  - Defining Your Widget Settings
  - Creating Your Widget Intent Receiver and Adding It to the Application Manifest
  - Introducing the App Widget Manager and Remote Views
  - Creating and Manipulating Remote Views
  - Applying Remote Views to Running App Widgets
  - Using Remote Views to Add Widget Interactivity
  - Refreshing Your Widgets
  - Using the Minimum Update Rate
  - Using Intents
  - Using Alarms
  - Creating and Using a Widget Configuration Activity
- Introducing Collection View Widgets

- Creating Collection View Widget Layouts
- Creating the Remote Views Service
- Creating a Remote Views Factory
- Populating Collection View Widgets Using a Remote Views Service
- Adding Interactivity to the Items Within a Collection View Widget
- Binding Collection View Widgets to Content Providers
- Refreshing Your Collection View Widgets

Introducing Live Folders
- Creating Live Folders
  - The Live Folder Content Provider
  - The Live Folder Activity

Surfacing Application Search Results Using the Quick Search Box
- Surfacing Search Results to the Quick Search Box

Creating Live Wallpaper
- Creating a Live Wallpaper Definition Resource
- Creating a Wallpaper Service
- Creating a Wallpaper Service Engine

Audio, Video, and using the Camera

Playing Audio and Video
- Introducing the Media Player
- Preparing Audio for Playback
  - Initializing Audio Content for Playback
- Preparing Video for Playback
  - Playing Video Using the Video View
  - Creating a Surface for Video Playback
- Controlling Media Player Playback
- Managing Media Playback Output
- Responding to the Volume Controls
- Responding to the Media Playback Controls
- Requesting and Managing Audio Focus
- Pausing Playback When the Output Changes
- Introducing the Remote Control Client

Manipulating Raw Audio
- Recording Sound with Audio Record
- Playing Sound with Audio Track

Creating a Sound Pool

Using Audio Effects

Using the Camera for Taking Pictures
- Using Intents to Take Pictures
- Controlling the Camera Directly
  - Camera Properties
  - Camera Settings and Image Parameters
  - Controlling Auto Focus, Focus Areas, and Metering Areas

        - Using the Camera Preview
        - Detecting Faces and Facial Features
        - Taking a Picture
    - Reading and Writing JPEG EXIF Image Details
- Recording Video
    - Using Intents to Record Video
    - Using the Media Recorder to Record Video
        - Configuring the Video Recorder
        - Previewing the Video Stream
        - Controlling the Recording
        - Creating a Time-Lapse Video
- Using Media Effects
- Adding Media to the Media Store
    - Inserting Media Using the Media Scanner
    - Inserting Media Manually

Bluetooth, NFC, Networks, and WI-FI

- Using Bluetooth
    - Managing the Local Bluetooth Device Adapter
    - Being Discoverable and Remote Device Discovery
        - Managing Device Discoverability
        - Discovering Remote Devices
    - Bluetooth Communications
        - Opening a Bluetooth Server Socket Listener
        - Selecting Remote Bluetooth Devices for Communications
        - Opening a Client Bluetooth Socket Connection
        - Transmitting Data Using Bluetooth Sockets
- Managing Network and Internet Connectivity
    - Introducing the Connectivity Manager
    - Supporting User Preferences for Background Data Transfers
    - Finding and Monitoring Network Connectivity
- Managing Wi-Fi
    - Monitoring Wi-Fi Connectivity
    - Monitoring Active Wi-Fi Connection Details
    - Scanning for Hotspots
    - Managing Wi-Fi Configurations
    - Creating Wi-Fi Network Configurations
- Transferring Data Using Wi-Fi Direct
    - Initializing the Wi-Fi Direct Framework
    - Enabling Wi-Fi Direct and Monitoring Its Status
    - Discovering Peers
    - Connecting with Peers
    - Transferring Data Between Peers
- Near Field Communication

Reading NFC Tags
Using the Foreground Dispatch System
Introducing Android Beam
Creating Android Beam Messages
Assigning the Android Beam Payload
Receiving Android Beam Messages
Telephony and SMS
Hardware Support for Telephony
Marking Telephony as a Required Hardware Feature
Checking for Telephony Hardware
Using Telephony
Initiating Phone Calls
Replacing the Native Dialer
Accessing Telephony Properties and Phone State
Reading Phone Device Details
Reading Network Details
Reading SIM Details
Reading Data Connection and Transfer State Details
Monitoring Changes in Phone State Using the Phone State Listener
Monitoring Incoming Phone Calls
Tracking Cell Location Changes
Tracking Service Changes
Monitoring Data Connectivity and Data Transfer Status Changes
Using Intent Receivers to Monitor Incoming Phone Calls
Introducing SMS and MMS
Using SMS and MMS in Your Application
Sending SMS and MMS from Your Application Using Intents
Sending SMS Messages Using the SMS Manager
Sending Text Messages
Tracking and Confirming SMS Message Delivery
Conforming to the Maximum SMS Message Size
Sending Data Messages
Listening for Incoming SMS Messages
Simulating Incoming SMS Messages in the Emulator
Handling Data SMS Messages
Emergency Responder SMS Example
Automating the Emergency Responder
Introducing SIP and VOIP
Advanced Android Development
Android Security
Linux Kernel Security
Introducing Permissions
Declaring and Enforcing Permissions

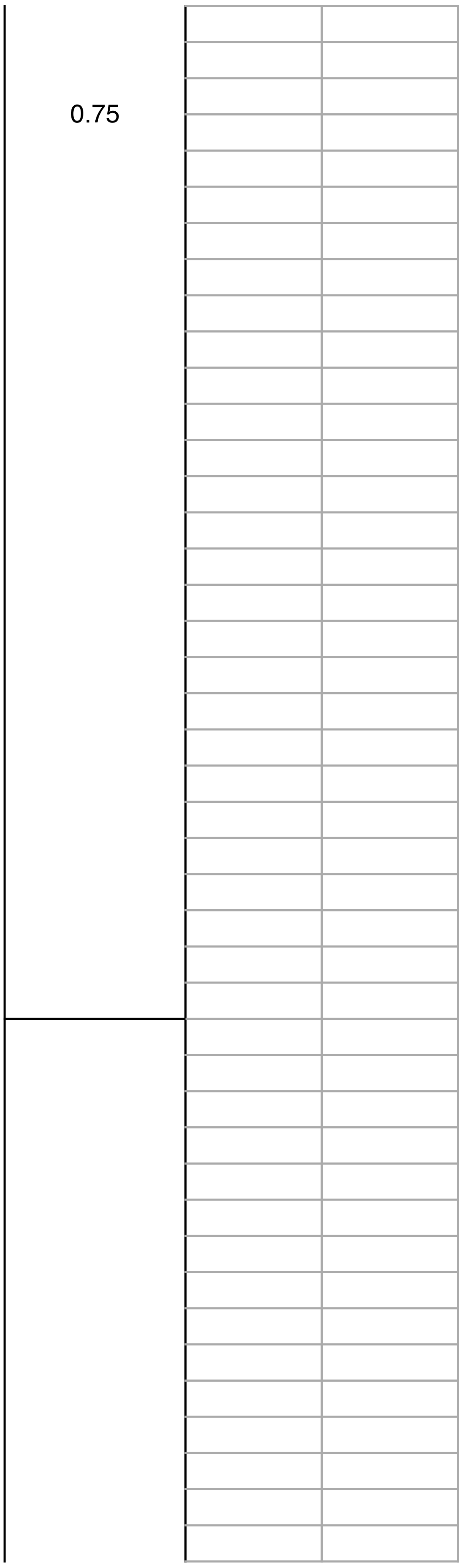
      - Enforcing Permissions when Broadcasting Intents
  - Introducing Cloud to Device Messaging (C2DM)
    - C2DM Restrictions
    - Signing Up to Use C2DM
    - Registering Devices with a C2DM Server
    - Sending C2DM Messages to Devices
    - Receiving C2DM Messages
  - Implementing Copy Protection Using the License Verification Library
    - Installing the License Verification Library
    - Finding Your License Verification Public Key
    - Configuring Your License Validation Policy
    - Performing License Validation Checks
  - Introducing In-App Billing
    - In-App Billing Restrictions
    - Installing the In-App Billing Library
    - Finding Your Public Key and Defining Your Purchasable Items
    - Initiating In-App Billing Transactions
    - Handling In-App Billing Purchase Request Responses
  - Using Wake Locks
  - Using AIDL to Support Inter-Process Communication for Services
    - Implementing an AIDL Interface
      - Making Classes Parcelable
      - Creating an AIDL Service Definition
      - Implementing and Exposing the AIDL Service Definition
  - Dealing with Different Hardware and Software Availability
    - Specifying Hardware as Required
    - Confirming Hardware Availability
    - Building Backward-Compatible Applications
      - Parallel Activities
      - Interfaces and Fragments
  - Optimizing UI Performance with Strict Mode

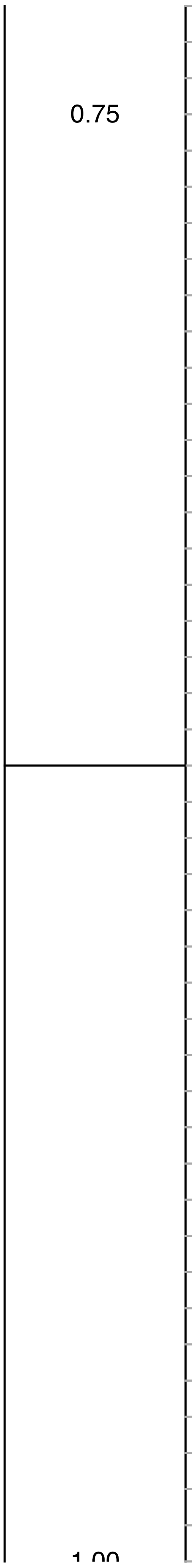
- **Monetizing, Promoting, and Distributing Applications**
  - Signing and Publishing Applications
    - Signing Applications Using the Export Android Application Wizard
  - Distributing Applications
    - Introducing Google Play
    - Getting Started with Google Play
    - Publishing Applications
    - Application Reports Within the Developer Console
    - Accessing Application Error Reports
  - An Introduction to Monetizing Your Applications
  - Application Marketing, Promotion, and Distribution Strategies
    - Application Launch Strategies

Promotion Within Google Play
Internationalization
Analytics and Referral Tracking
Using Google Analytics for Mobile Applications
Referral Tracking with Google Analytics

Back		
Duration (Days)		
0.75		

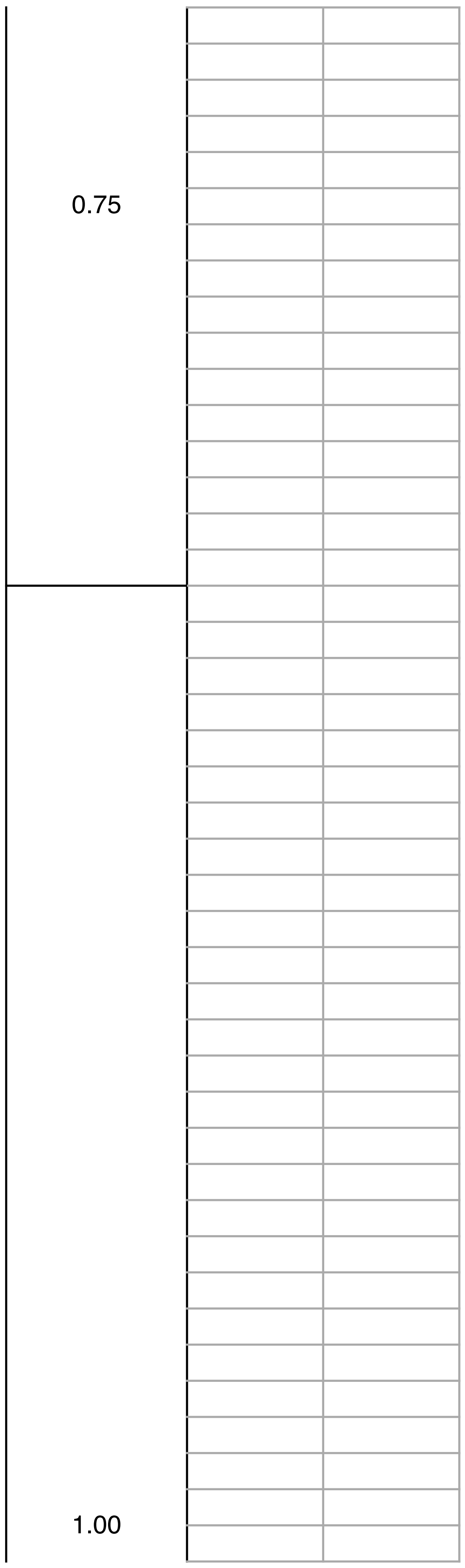
0.75


0.75


0.75
1.00

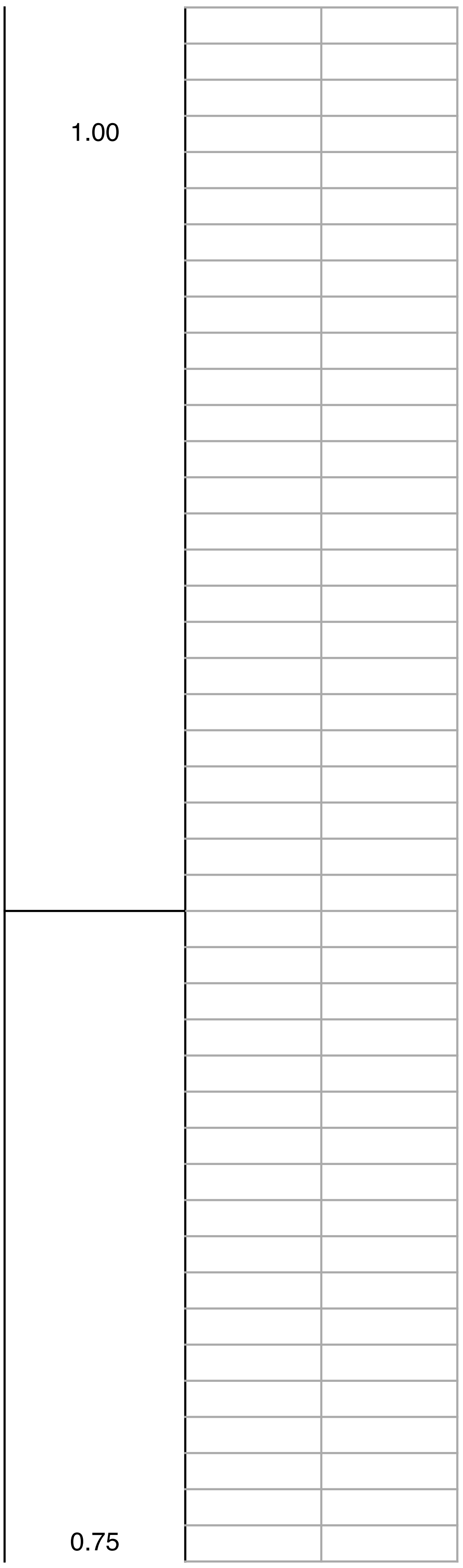
1.00

1.00


0.75
1.00

1.00		

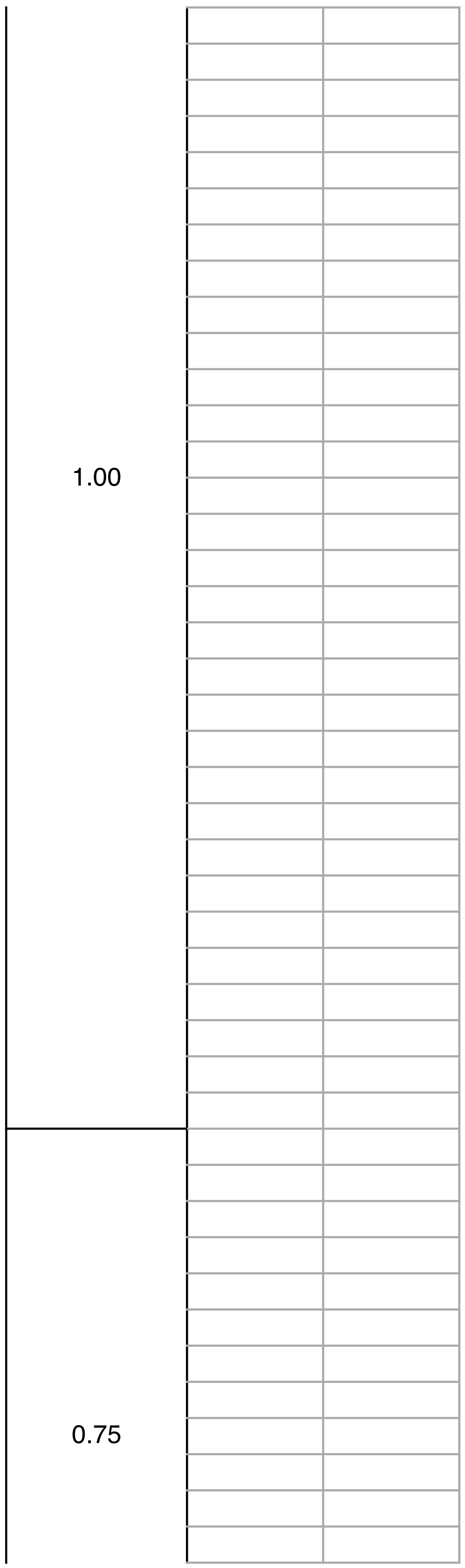
0.75		


1.00
0.75

1.00

1.00

1.00		


1.00
0.75

Project Based Learning Scenario

Assignment 1

Project: Develop a Vehicle Insurance Claim Mobile App
Project Overview:
Students will develop a mobile application that streamlines the vehicle insurance claim process. The a
will allow users to file claims, upload damage photos, track claim status, and communicate with insurance agents. This project will incorporate various Android development skills, focusing on user interface design, data management, and integrating hardware features.

Detailed Description with User Stories:
User Story 1: As a user, I want to set up my profile and link it to my insurance policy so that I can quickly file claims when needed.

User Story 2: As a user, I want to easily navigate through the app to find the features I need without confusion.

User Story 3: As a user, I want to use my camera to take pictures of the vehicle damage and upload them directly to my claim file.

User Story 4: As a user, I want my data to transfer securely and quickly when I submit a claim or communicate with my insurance agent.

User Story 5: As a user, I want to find authorized service centers or garages near me using the app's map feature.

User Story 6: As a user, I want to view my claim history and status updates for my ongoing claims.

User Story 7: As a developer, I want to create an efficient and responsive app that adapts to various devices and screen sizes.

User Story 8: As a developer, I want to build a prototype of the insurance claim app to test its functionality and gather user feedback.

User Story 9: As a user, I want an intuitive user interface that makes filing a claim or checking my clai
status easy.

User Story 10: As a developer, I want to use fragments to create a modular and adaptable user interface for the app.

User Story 11: As a user, I want to receive real-time notifications about the status of my insurance claim.

User Story 12: As a developer, I want the app to access and display relevant information from the wel
like updates from the insurance company.

User Story 13: As a user, I want the app to process tasks in the background, such as updating the cla
status, without interrupting my use of the app.

User Story 14: As a user, I want a refined user experience that includes features like custom dialogs f
confirming claim submissions and full-screen photos of my vehicle damage.

User Story 15: As a user, I want to record a video statement about the accident and upload it to my claim file.

User Story 16: As a user, I want the app to maintain a stable connection whether I'm at home or on th
go to ensure I can manage my claims anytime.

Module
Duration (Days
Sl. No.
1.0
1.1
1.2
2.0
2.1
2.2
3.0
3.1
3.2
4.0

4.1
4.2
4.3
5.0
5.1
5.2
5.3
6.0
6.1
6.2

6.2
7.0
7.1
8.0
8.1
8.2

Kotlin Programming
4
Topics
Kotlin Introduction and Setup
Kotlin Introduction
Overview of Kotlin and its position in modern development.
History of Kotlin
Kotlin's advantages over other languages.
Basic Syntax and Setup
Setting up the environment and understanding basic syntax.
Installing Kotlin
Setting up an IDE (e.g., IntelliJ IDEA)
Writing and running a basic Kotlin program.
Variables and Control Structures
Variables and Data Types
Defining and using variables
Variable declaration and initialization
Variable naming conventions
Understanding basic data types.
Integers, floating-point numbers, characters, booleans, and strings
Control Structures: If and When
Using conditional statements
If expressions with examples
Conditional operators (e.g., &&, II)
When clauses.
Pattern matching with when
Multiple conditions in when.
Functions and OOP Basics
Type Inference and Type Checking
How Kotlin infers variable types
Type inference examples
Explicit type declarations.
Declaring variable types explicitly
Functions: Declaration and Usage
Defining functions
Function declaration syntax
Function arguments
Passing arguments to functions
Return types.
Specifying return types in functions
Advanced OOP and Data Handling

Lambdas and Higher-Order Functions
Lambda expressions
Lambda syntax and examples
Higher-order functions and function types.
Using higher-order functions
Function types in Kotlin
Classes: Basics and Constructors
Creating classes and objects
Class declaration and object instantiation
Primary and secondary constructors
Constructors and their usage
Inheritance and Class Hierarchies
Extending classes
Inheriting properties and methods
Overriding methods and properties.
Method and property overriding in subclasses.
Inheritance, Interfaces, and Advanced Concepts
Interfaces and Abstract Classes
Defining interfaces
Interface declaration and implementation
Implementing multiple interfaces.
Implementing multiple interfaces in a class
Visibility Modifiers and Encapsulation
Public, private, protected, internal
Different visibility modifiers and their usage
Encapsulation and getters/setters.
Encapsulation principles
Using getters and setters in Kotlin
Data Classes and Sealed Classes
Creating data classes
Data class syntax
Using sealed classes for restricted hierarchies.
Sealed class declaration
Creating subclasses of sealed classes.
Data Classes, Generics, and Collections
Generics and Type Variance
Generic classes and functions
Defining generic functions and classes
Understanding type variance in Kotlin.
Covariance and contravariance
Collections: Lists, Sets, Maps
Using Kotlin's collection libraries
Creating and initializing lists, sets, and maps

Iterating and modifying collections.
Looping through collections
Adding, removing, and updating elements in collections.
Null Safety and Exception Handling
Null Safety and Optional Handling
Handling null values with safe calls and elvis operator.
Safe call operator (?.)
Elvis operator (?:)
Exception Handling and Try-Catch
Handling exceptions with try-catch blocks
Custom exception classes.
Advanced Kotlin Features
Extensions: Functions and Properties
Extending existing classes with functions and properties.
Creating extension functions
Adding extension properties
Coroutines: Basics and Usage
Introduction to coroutines for managing asynchronous tasks.
Coroutines vs. threads
Coroutine builders (e.g., launch, async)
Asynchronous Programming Patterns
Structured concurrency
Suspended functions and async/await.

Back

Remarks	Duration (Days)
Learners will get a good grasp of Kotlin, its benefits, and its role in modern development. They'll set up their development environment, learn basic syntax, and run Kotlin programs. Understanding variables, data types, and control structures like if and when statements is important for writing good Kotlin code, ensuring they can create efficient and reliable applications. This foundational knowledge is crucial for building quality apps with Kotlin.	0.25
	0.5
Learners will acquire a solid understanding of fundamental programming concepts. They will explore type inference and checking, grasping how Kotlin infers variable types and the use of explicit type declarations. The course covers functions comprehensively, from declaration to usage, including arguments and return	0.5

types. Learners will delve into advanced object-oriented programming (OOP) topics such as lambdas, higher-order functions, and class creation with constructors. They will also master inheritance, class hierarchies, and method/property overriding. This knowledge forms a strong foundation for effective Kotlin programming, enabling the development of robust and efficient applications in this modern programming	0.75	
Learners will delve into advanced Kotlin concepts, including inheritance, interfaces, and encapsulation. They'll understand interface declaration and implementation, including implementing multiple interfaces in a class. Visibility modifiers like public, private, protected, and internal will be covered, along with encapsulation principles and the use of getters and setters. Additionally, they'll explore data classes and sealed classes, learning how to create them and their syntax. Sealed classes will be used for defining restricted hierarchies, including the creation of subclasses. These advanced concepts enhance their ability to build robust and maintainable Kotlin applications.	0.5	
Learners will dive into essential topics of data classes, generics, and collections in Kotlin. They will grasp the concept of generics, enabling them to define generic classes and functions. Understanding type variance, including covariance and contravariance, will be covered. In addition, learners will explore Kotlin's collection libraries, gaining the ability to create, initialize, iterate, and modify lists, sets, and maps efficiently. These foundational concepts equip them with the skills	0.5	

to work with data, generics, and collections effectively in Kotlin development.		
Learners will master null safety and exception handling in Kotlin. They'll handle null values using safe calls and the Elvis operator (?. and ?:). Additionally, they will understand exception handling with try-catch blocks, including creating custom exception classes for more robust error management. These skills ensure safer and more reliable Kotlin code.	0.25	
Learners will explore advanced Kotlin features, including extensions and coroutines. They'll extend existing classes with custom functions and properties, learning to create extension functions and properties. Additionally, they'll dive into coroutines, understanding their benefits over traditional threads, coroutine builders like launch and async, and asynchronous programming patterns such as structured concurrency and suspended functions with async/await. These advanced features empower them to write more efficient and responsive Kotlin code.	0.75	

Project Based Le

Assignment 1

Development Scenario 1: Personal Finance Tracker

Day 1: Introduction and Setup and Variables and Control Structures
Task 1: Install Kotlin and configure IntelliJ IDEA. Verify the setup by running a "Hello, World!" program.
Task 2: Explore Kotlin REPL (Read-Eval-Print Loop) to familiarize with Kotlin syntax and basic operations.
Task 3: Create a Transaction class with properties such as amount, date, and category.
Task 4: Implement control structures to categorize transactions (e.g., Food, Utilities, Entertainment) using when statements.

Day 2: Functions and OOP Basics
Task 5: Write functions to add, delete, and edit transactions in a TransactionList class.
Task 6: Develop a simple User class with methods to login and display a summary of expenses.
Task 7: Use lambdas and higher-order functions to filter and sort transactions by date or amount.
Task 8: Implement inheritance by creating specific transaction classes like Income and Expense that inherit from Transaction.

Day 3: Interfaces, Encapsulation, and Advanced Concepts / Collections and Generics
Task 9: Define an Exportable interface with a method to export transaction data to CSV.
Task 10: Apply encapsulation to Transaction properties using getters and setters ensuring sensitive data is protected.
Task 11: Create generic functions to handle different types of collections (List, Set, Map) of transactions.
Task 12: Utilize Kotlin's collection libraries to manage a collection of User objects, enabling the addition and removal of users.

Day 4: Null Safety and Exception Handling / Advanced Features (Extensions and Coroutines)
Task 1: Implement null safety features to handle the absence of transaction data.
Task 2: Write custom exception classes to handle errors related to transaction processing.
Task 3: Create extension functions for the List<Transaction> class to calculate total expenses and incomes.
Task 4: Use coroutines to handle simultaneous processing of importing and exporting transaction data without blocking the main thread.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.

2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.
3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day

earning Scenario

Assignment 2

Development Scenario 2: Event Management System

Day 1: Introduction and Setup
Task 1: Set up the Kotlin development environment and write a simple Kotlin script to validate the setup.
Task 2: Experiment with Kotlin's string templates to create dynamic welcome messages.
Task 3: Define data types to represent event details such as name, date, and attendee count.
Task 4: Implement a basic user input flow to create new events using if and when statements.

Day 2: Functions and OOP Basics
Task 5: Design a EventManager class with methods to add and remove events.
Task 6: Create a Display interface with a method to show event details and implement it in the EventManager.
Task 7: Utilize higher-order functions to implement a simple notification system for event updates.
Task 8: Construct subclass SpecialEvent with additional features like VIP lists and premium services.

Day 3: Interfaces, Encapsulation, and Advanced Concepts / Collections and Generics
Task 9: Develop a Schedule class that uses interfaces to ensure that all event types can be scheduled and rescheduled.
Task 10: Secure the event data with proper encapsulation and visibility modifiers.
Task 11: Manage a collection of events allowing filtering by date or type using Kotlin's powerful collection operations.
Task 12: Use generics to create a flexible DataManager class capable of handling different data types, including attendees and events.

Day 4: Null Safety and Exception Handling/Advanced Features (Extensions and Coroutines)
Task 1: Ensure that the system gracefully handles null references when retrieving event data.
Task 2: Implement try-catch blocks to handle parsing errors when reading event dates and times.
Task 1: Write extension functions for the Event class to add features like tagging and categorization.
Task 2: Introduce coroutines to concurrently handle event bookings and cancellations.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing

tool.
3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day

Module
Duration (Days
Sl. No.
1.0
1.1
1.2
1.3
1.4
2.0
2.1
2.2
3.0
3.1
3.2

4.0
4.1
4.2
5.0
5.1
5.2
5.3
5.4
5.5
5.6
6.0
6.1
6.2
6.3
7.0
7.1
7.2
7.3
7.4
7.5
8.0
8.1
8.2
8.3

9.0
9.1
9.2
9.3
10.0
10.1
10.2
10.3
11.0
11.1
11.2
11.3
12.0
12.1
12.2
12.3
13.0
13.1
13.2
13.3
13.4
13.5
14.0
14.1
14.2
14.3

Android Framework with Kotlin
7.00
Topics
Android Introduction and Setup
Android Ecosystem and Architecture
Overview of Android OS and its versions
Android architecture components.
Android Studio and Development Tools
Using Android Studio for app development
Creating a new Android project
Navigating the Android Studio interface
Overview of Android Studio features
Debugging, profiling, and emulator.
Project Structure and UI Design
Project Structure and Gradle Build System
Exploring Android project components
Manifest file, Java/Kotlin files, XML layout files
Configuring Gradle build files.
Adding dependencies
Customizing build.gradle.
Activity Lifecycle and State Management
Understanding Activities
Activity lifecycle states (e.g., onCreate, onResume)
Lifecycle events
Implementing lifecycle callbacks
Managing activity state.
Saving and restoring instance state
Handling configuration changes.
UI Enhancements and Data Binding
View Binding and Data Binding
Using View Binding
Enabling View Binding in Android projects
Accessing UI elements using View Binding
Using Data Binding for efficient UI code.
Binding data to XML layouts
Data Binding expressions.
RecyclerView for Efficient List Handling
Creating lists and grids with RecyclerView
RecyclerView setup and layout managers
Custom RecyclerView adapters.
Creating custom adapter classes

Binding data to RecyclerView.
User Interaction, Navigation, and Data Handling
Handling User Inputs and Events
Capturing and handling user interactions and events.
Handling button clicks, text input, etc.
Implementing user interface feedback.
Navigation Component and Navigation Graph
Implementing app navigation using Navigation Architecture Component
Creating a navigation graph
Defining destinations and actions
Building a navigation graph.
Navigating between destinations.
Networking
Making Network Requests with Retrofit
Retrofit Setup and Configuration
Retrofit library setup in Android projects.
Configuration of Retrofit for API calls.
Creating API Service Interfaces
Handling Network Responses
Processing responses from network requests.
Handling different HTTP response codes.
Parsing JSON Data
Parsing JSON data received from the server.
Error Handling and Response Handling
Handling errors and exceptions in network operations.
Coroutines for Asynchronous Tasks
Handling Asynchronous Operations with Kotlin Coroutines
Launching Coroutines
Starting and managing Kotlin coroutines.
Suspending Functions and Asynchronous Programming
Use of suspending functions for asynchronous tasks.
WorkManager for Background Tasks
Scheduling Background Tasks Efficiently with WorkManager
Defining and Scheduling One-time and Periodic Tasks
Configuration and scheduling of tasks with WorkManager.
Managing and Monitoring Background Tasks
Tracking the progress and status of background tasks.
Ensuring Task Execution
Handling Task Failures
Dependency Injection with Dagger/Hilt
Implementing Dependency Injection
Setting Up Dagger/Hilt in Android Projects
Injecting Dependencies into Activities and Fragments

Google Maps Integration
Integrating Google Maps for Location-based Services
Adding a Google Map Fragment to an App
Handling User Location and Markers
Firebase Integration for Backend Services
Adding Backend Functionalities with Firebase Services
Firebase Authentication
Firestore Database Integration
Material Design Guidelines and Components
Applying Material Design Principles for UIs
Implementing Material Design Components
Usage of Material Design components.
Creating Responsive and User-friendly UIs
Testing: Unit Tests and Instrumentation Tests
Writing and running unit tests and UI tests for robust apps.
JUnit and Espresso for testing
Test-driven development (TDD).
Performance Optimization and Memory Management
Identifying and solving performance issues
Profiling app performance
Managing app memory efficiently.
Avoiding memory leaks
Using the Android Memory Profiler.
Publishing and Distributing Android Apps
Preparing and publishing apps on the Google Play Store.
App signing and release management
Google Play Store guidelines and policies.

Back		
Duration (Days)		
0.75		
0.75		
1.25		

1.00		
0.75		

1.50		
1.00		

Project Based Le

Assignment 1

Development Scenario 1: Smart Home Controller App
Day 1: Android Introduction and Setup
Task 1: Install the necessary SDKs and emulators for the Smart Home Controller project.
Task 2: Create the initial project structure with a basic activity.

Day 2: Project Structure and UI Design
Task 1: Construct the MainActivity layout with controls for smart devices (like lights, thermostats).
Task 2: Set up a Gradle configuration for dependency management and build customization.

Day 3: UI Enhancements and Data Binding
Task 1: Utilize Data Binding to connect UI components with the backend logic.
Task 2: Create a dashboard UI showing the status of various smart devices using data binding.

Day 4: User Interaction, Navigation, and Data Handling
Task 1: Implement event handling for user interactions with device controls.
Task 2: Develop a settings screen for device preferences and configure navigation to this screen.

Day 5: Networking
Task 1: Set up Retrofit to communicate with a backend server managing smart devices.
Task 2: Handle server responses, updating the UI accordingly, and manage potential network errors.

Day 6: Coroutines for Asynchronous Tasks
Task 1: Use Kotlin coroutines to perform network operations for sending device control commands.
Task 2: Apply suspending functions to update device statuses in real-time.

Day 7: WorkManager for Background Tasks
Task 1: Implement WorkManager to periodically sync device states with the server.
Task 2: Use WorkManager to schedule tasks for device automation based on user-defined rules

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.
3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day.

earning Scenario

Assignment 2

Development Scenario 2: Vehicle Health Monitoring System
Day 1: Android Introduction and Setup
Task 1: Install the necessary SDKs and emulators that simulate vehicle hardware interfaces.
Task 2: Initialize the Android project with a basic activity that will serve as the dashboard for vehicle health metrics.

Day 2: Project Structure and UI Design
Task 1: Design the MainActivity layout to display real-time data such as engine health, oil level, and tire pressure.
Task 2: Configure Gradle for dependency management, focusing on libraries necessary for vehicle data communication.

Day 3: UI Enhancements and Data Binding
Task 1: Implement Data Binding to connect the dashboard UI components to the backend vehicle data.
Task 2: Develop a dynamic UI that reflects the vehicle's health status, using LiveData to ensure real-time updates.

Day 4: User Interaction, Navigation, and Data Handling
Task 1: Create interactive elements for users to report issues or request maintenance services.
Task 2: Build a settings screen where users can set thresholds for alerts (like minimum tire pressure).

Day 5: Networking
Task 1: Establish Retrofit communication with a backend server for storing long-term vehicle health data and scheduling services.
Task 2: Ensure the network layer gracefully handles errors, with retry mechanisms and user notifications for connectivity issues.

Day 6: Coroutines for Asynchronous Tasks
Task 1: Use Kotlin coroutines for handling asynchronous communications with onboard diagnostics and sensors.
Task 2: Implement functions that use coroutines to fetch and push data to the backend without blocking the user interface.

Day 7: WorkManager for Background Tasks
Task 1: Integrate WorkManager to conduct daily diagnostics and sync this data with the user's account on the server.
Task 2: Set up WorkManager to remind users of upcoming maintenance or service appointments based on vehicle usage data and manufacturer recommendations.

Submission Guidelines:
Ensure clarity and conciseness, demonstrating understanding of Android development and vehicle monitoring concepts.

Include diagrams of system architecture and data flow, which can be created digitally or by hand.
Cite any external sources used for developing the system.
Compile the entire project documentation into a single PDF and submit the code to GitLab by the deadline.

Module
Duration (Days
Sl. No.
1.0
1.1
1.0
2.0
2.1
2.2
3.0
3.1
3.2
4.0
4.1
4.2
5.0
5.1
5.2

6.0
6.1
6.2
6.3
7.0
7.1
7.2
8.0
8.1
8.2
8.3
9.0
9.1
9.2
10.0
10.1

11.0
11.1
12.0
12.1
12.2
13.0
13.1
13.2
14.0
14.1
14.2
15.0
15.1
15.2
16.0
16.1

16.2
16.3
17.0
17.1
17.2
17.3
18.0
18.1
18.2
18.3
19.0
19.1
20.0
20.1
21.0
21.1
21.2

22.0
22.1
22.2
22.3
22.4
22.5

iOS with Swift, SwiftUI Development
18.00
Topics
iOS Introduction and Setup
iOS Platform Overview and Ecosystem
Introduction to the iOS platform.
History and the Apple ecosystem.
Xcode IDE and Interface Builder
Using Xcode for iOS development.
Interface Builder for UI design.
Introduction to Swift and Setup
Introduction to Swift and Its Evolution
Overview of Swift language.
History and modern applications of Swift.
Swift Environment Setup and Basics
Setting up the development environment.
Understanding Swift syntax.
Variables, Constants, and Control Flow
Variables, Constants, and Data Types
Use of variables and constants.
Understanding different data types in Swift.
Integers, floating-point numbers, characters, booleans, strings.
Control Flow: If-Else, Switch, Loops
Implementing control structures.
If-else statements.
Switch statements.
Loops (for, while, repeat-while).
Functions and Closures
Functions: Declaration and Use
Defining and calling functions.
Understanding function parameters and return types.
Closures: Basics and Usage
Creating and using closures.
Using closures as a way to pass executable code.
OOP with Classes, Structures, and Enumerations
Classes, Structures, and Enumerations
Exploring object-oriented programming.
Working with classes, structs, and enums.
Properties and Methods in Swift
Working with properties.
Defining methods and computed values.

Advanced OOP and Error Handling
Inheritance and Polymorphism
Implementing inheritance.
Subclassing and polymorphism concepts.
Protocols and Extensions
Using protocols for delegation.
Extensions for adding functionality.
Error Handling in Swift
Managing errors and exceptions.
Swift's error handling model.
Dependency Injection
Understanding Dependency Injection
Why use dependency injection?
Types of dependency injection: constructor, property, and method injection
Understanding dependency inversion and its relation to dependency injection
Dependency Injection Techniques in Swift
Building a simple DI container
Implementing constructor injection in Swift
Using property wrappers for dependency injection
Overview of common Swift dependency injection frameworks (e.g., Swinject, Resolver)
Optionals, Collections, and Generics
Optionals and Unwrapping
Working with optionals.
Safely unwrapping optional values.
Collections: Arrays, Sets, Dictionaries
Using collections.
Arrays, sets, and dictionaries for data storage.
Swift Generics and Type Safety
Creating generic functions and types.
Ensuring type safety in code.
Advanced Features and Memory Management
Advanced Features: Tuples, Guards, Defer
Exploring advanced Swift features.
Tuples and their usage.
Guard statements for early exit.
Defer for cleanup operations.
Memory Management and ARC
Understanding automatic reference counting (ARC).
Managing memory efficiently.
SwiftUI Basics and UI Building
SwiftUI Basics and Interface Components
Introduction to SwiftUI.
Declarative syntax for UI building.

Creating UI Elements
Building UI elements like Text, Image, Button, and TextField.
SwiftUI Layout and Views
SwiftUI Layout System and Views
Exploring SwiftUI's layout system.
Working with views and modifiers.
Customizing Views
Applying modifiers for customization.
Combining views and modifiers to create complex UIs.
User Interaction and Navigation
Handling User Interactions in SwiftUI
Implementing user interaction handling in SwiftUI apps.
Navigation and Passing Data in SwiftUI
Creating navigational flows.
Passing data between views in SwiftUI.
Concurrency and Advanced Topics
Concurrency: Threads and GCD
Managing concurrency with threads.
Grand Central Dispatch (GCD) for asynchronous programming.
Pattern Matching and Enums with Associated Values
Using pattern matching for value extraction.
Enums with associated values for complex data modeling.
Data Binding, State Management and Combine Framework
State and Data Binding in SwiftUI
Managing app state.
The role of data bindings in decoupling application logic from UI
The @Binding property wrapper and its use cases
Implementing two-way bindings in SwiftUI forms
Using data binding for reactive UI updates.
Using Combine and Observables in SwiftUI
Integrating Combine framework for handling asynchronous events.
Combining publishers and subscribers for data flow.
Lists, Animation, and Adaptivity
List and Collection Views in SwiftUI
Building dynamic lists and collections in SwiftUI.
Customizing list layouts and styles.
Animation and Gestures in SwiftUI
Implementing animations and transitions.
Handling gestures for interactive UI.
Advanced SwiftUI Topics
SwiftUI for Adaptive UIs and Dark Mode
Designing adaptive UIs that support multiple screen sizes.
Implementing Dark Mode support.

Integrating Core Data with SwiftUI
Using Core Data for persistent data storage in SwiftUI apps.
Fetching, saving, and updating Core Data entities.
SwiftUI and RESTful API Integration
Connecting SwiftUI apps with web services.
Fetching and displaying data from RESTful APIs.
Widgets, Testing, and Performance
SwiftUI and WidgetKit for Home Screen Widgets
Developing customizable widgets for the iOS home screen with WidgetKit.
Configuring widget previews and data.
Testing SwiftUI Apps with XCTest
Writing unit tests and UI tests for SwiftUI applications.
Using XCTest for testing SwiftUI views and logic.
SwiftUI Performance Tuning and Debugging
Optimizing SwiftUI app performance.
Effective debugging strategies and tools.
App Life Cycle, Notifications, and Hardware Access
App Life Cycle and Scene Management
Understanding the iOS app life cycle.
Managing app scenes for multitasking.
Push Notifications and Local Notifications
Configuring and handling push and local notifications in iOS apps.
Scheduling and displaying notifications.
Camera and Photo Library Access in iOS
Accessing and using the camera and photo library in iOS applications.
Capturing photos and videos, selecting images from the library.
Swift Application Packaging and Management
Swift Package Manager and Dependency Management
Managing external libraries and dependencies.
Utilizing Swift Package Manager for package management.
App Deployment and Conclusion
Publishing iOS Apps to the App Store
Preparing app for submission to the Apple App Store.
Configuring app information, icons, and screenshots.
Solid Principle
Overview of SOLID Principles
Importance of SOLID principles in software design
Detailed exploration of each principle with Swift examples
Applying SOLID Principles in iOS Development
Single Responsibility Principle: Ensuring a class has only one reason to change
Open/Closed Principle: Extending a class's behavior without modifying it
Liskov Substitution Principle: Substituting a subclass object with a superclass object
Interface Segregation Principle: Creating fine-grained interfaces for client-specific needs

Dependency Inversion Principle: Decoupling high-level modules from low-level modules
Practical exercises: Identifying SOLID violations and refactoring an iOS app’s codebase
Introduction to Design Patterns
The concept of design patterns in software engineering
The Importance of Design Patterns in Software Development
Categories of Design Patterns
Design Patterns in Swift and iOS Development
Creational Patterns
Singleton: Purpose and Implementation in Swift
Builder: Building Complex Objects
Factory Methods: Encapsulation of Object Creation
Structural Patterns
Adapter: Bridging Between Interfaces
Facade: Simplifying Complex Systems
Behavioral Patterns
Observer: Reactive Programming with Notifications
Command: Encapsulating Invocation Information
Architectural Patterns in iOS
MVC (Model-View-Controller) in Swift
MVVM (Model-View-ViewModel) with Swift and SwiftUI
The Model-View-ViewModel (MVVM) architecture
Comparison with MVC (Model-View-Controller)
Responsibilities and lifecycle of a ViewModel in iOS
Converting an MVC example to MVVM

Back

Remarks
Learners will gain a foundational understanding of iOS, its ecosystem, and the essential tools like Xcode and Interface Builder. This knowledge is crucial for anyone looking to develop iOS applications.
Learners will gain proficiency in Swift programming, essential for developing iOS and macOS applications. This knowledge is necessary for anyone aspiring to create software in the Apple ecosystem, providing the foundation needed to build functional and user-friendly applications for various Apple devices. They will learn Swift's syntax, variables, data types, and control flow, ensuring a solid understanding of Swift fundamentals for app development.
Learners will delve into Swift's functions and closures, mastering the declaration and use of functions with parameters and return types. They will also explore closures, understanding their basics and how to use them effectively. They will learn object-oriented programming with classes, structures, and enumerations, including properties and methods. Advanced topics include inheritance, polymorphism, protocols for delegation, extensions for added functionality, and Swift's error handling model, providing a comprehensive understanding of Swift programming for app development and error management.

Dependency Injection in Swift explores its purpose for decoupling, various injection methods—constructor, property, method—and inversion principle. It covers practical DI techniques, including container creation, constructor injection, property wrappers, and popular frameworks like Swinject and Resolver.
learner will gain knowledge on how to work with advanced features and safely unwrap optional values. It also explores collections like arrays, sets, and dictionaries for efficient data storage and retrieval. Additionally, they will learn Swift generics, enabling them to create generic functions and types while ensuring type safety in their code, a crucial aspect of robust software development in Swift.
Learners will explore advanced Swift features, including tuples for organizing and working with multiple values, guard statements for early exit from code blocks, and defer for performing cleanup operations. Thewill learn memory management in Swift, focusing on automatic reference counting (ARC) to ensure efficient memory usage and avoid memory leaks, a critical aspect of Swift app development.
Learner will delve into SwiftUI, learning its declarative syntax for building user interfaces and creating various UI elements. This knowledge is crucial for anyone looking to develop iOS applications, as it forms the basis for designing and building user-friendly apps in the Apple ecosystem.

Learners will delve into SwiftUI's layout system, gaining expertise in working with views and modifiers to create customized user interfaces. They will also learn to handle user interactions and implement navigation within SwiftUI apps, including passing data between views. This knowledge is essential for developing interactive and user-friendly iOS applications using SwiftUI.
Learners will delve into concurrency, covering the management of threads and asynchronous programming using Grand Central Dispatch (GCD) for efficient parallel processing. Additionally, they will be introduced to SwiftUI, a powerful tool for creating user interfaces, and explores SwiftUI previews for live view previews during development. Lastly, learners will discover pattern matching techniques and the use of enums with associated values for versatile data modeling.
Learners will know state management in SwiftUI, to effectively manage app state and utilize data binding for responsive UI updates. They will learn Combine framework, how to integrate it into SwiftUI for handling asynchronous events and implementing efficient data flow within iOS applications. These skills are crucial for building dynamic and responsive user interfaces.
Learners will master dynamic list and collection view creation in SwiftUI, enabling them to craft versatile user interfaces with customizable layouts. Additionally, they will acquire skills in implementing animations, transitions, and gesture handling for interactive and visually engaging app experiences. These capabilities are crucial for creating exceptional iOS applications.
Learners will delve into advanced SwiftUI topics, including the creation of adaptive user interfaces for different screen sizes and implementing Dark Mode support. They will also gain expertise in integrating Core Data for data storage and connecting SwiftUI apps with RESTful APIs for dynamic data retrieval, enhancing their ability to build versatile and data-driven iOS

applications.
Learners will explore the creation of iOS home screen widgets using SwiftUI and WidgetKit, allowing them to develop custom widgets for enhanced user experiences. They will also master the art of testing SwiftUI apps using XCTest, ensuring the reliability and quality of their applications. Additionally, they will discover performance tuning and debugging techniques to optimize SwiftUI app performance and resolve issues effectively.
Learners will delve into the intricacies of the iOS app life cycle and scene management, gaining a comprehensive understanding of how apps behave on iOS devices. They will also learn to implement push and local notifications, enhancing user engagement and interaction. Additionally, they will acquire the skills to access and utilize device hardware, such as the camera and photo library, enabling them to create feature-rich applications.
Learners will explore Swift Package Manager for efficient management of external libraries and dependencies, streamlining package management in Swift applications.
Learners will explore how to publish iOS applications to the Apple App Store. They will learn to prepare their apps for submission, configure essential app details, icons, and screenshots,for a successful deployment.
The session provides an in-depth look at SOLID principles in iOS development, explaining the importance of single responsibility, open/closed, Liskov substitution, interface segregation, and dependency inversion. It includes detailed Swift examples for each principle and practical exercises to identify and correct SOLID violations, enhancing codebase quality and maintainability in iOS app development.

In this session, learners will explore the fundamental design patterns critical for efficient iOS development with Swift. They'll understand the role of patterns in software engineering and their impact on creating robust, scalable apps. The course covers creational patterns for object creation, structural patterns to bridge interfaces, and behavioral patterns for enhancing inter-object communication. Learners will also delve into iOS-specific architectural patterns, comparing MVC with the more dynamic MVVM, understanding their usage, and applying them in real-world scenarios. This knowledge is essential for writing maintainable code and building sophisticated applications that stand the test of time.

Duration (Days)		
0.50		
0.50		
0.75		
0.75		
0.75		

0.75		
0.75		
0.75		
0.75		
0.75		

0.75		
1.00		
1.00		
1.00		

1.00		
1.50		
1.00		
0.50		
0.50		
0.75		

2.00		

Project Based Le

Assignment 1

Development Scenario 1: Personal Budget Tracker

Day 1: iOS Introduction and Setup
Task 1: Configure the Xcode development environment and create a new project for the budget tracker app.
Task 2: Familiarize with the Swift language by creating basic data models for expenses and income.

Day 2: SwiftUI Basics and UI Building
Task 1: Design the main user interface using SwiftUI to display a list of transactions.
Task 2: Implement a form to add new transactions, categorizing them as either income or expenses.

Day 3: SwiftUI Layout and Views
Task 1: Use SwiftUI's layout system to create a dashboard view that shows the user's current balance and spending trends.
Task 2: Integrate SwiftUI modifiers to customize the look and feel of the dashboard according to user preferences.

Day 4: User Interaction and Navigation
Task 1: Develop interaction logic that allows users to edit or delete transactions from the list.
Task 2: Set up navigation flows to move between the dashboard, transaction list, and transaction creation form.

Day 5: State Management and Combine Framework
Task 1: Implement state management using the Combine framework to update the user interface reactively as transactions are added or modified.
Task 2: Create publishers and subscribers that handle the loading and saving of transaction data in real-time.

Day 6: Lists, Animation, and Adaptivity
Task 1: Build dynamic lists that display transactions with animations to visually distinguish between income and expenses.
Task 2: Implement gesture handling to interact with list items, such as swipe to delete.

Day 7: Advanced SwiftUI Topics and RESTful API Integration
Task 1: Add support for adaptive UIs to ensure the app looks great on all device sizes, including Dark Mode support.
Task 2: Integrate with a RESTful API to fetch real-time currency exchange rates to adjust the values of transactions made in foreign currencies.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.

3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day.

earning Scenario
Assignment 2

Development Scenario 2: Medication Reminder App
Day 1: iOS Introduction and Setup
Task 1: Set up Xcode and review the Swift language basics, focusing on syntax and control flow.
Task 2: Start a new project for a medication reminder app, establishing the basic navigation and view structure.

Day 2: SwiftUI Basics and UI Building
Task 1: Develop the user interface components for creating medication reminders, such as name, dosage, and frequency.
Task 2: Create a SwiftUI view that lists all active medication reminders.

Day 3: SwiftUI Layout and Views
Task 1: Apply layout and views in SwiftUI to design a detailed medication info card that shows next dose time and remaining pills.
Task 2: Use view modifiers to style the medication info cards and make them user-friendly.

Day 4: User Interaction and Navigation
Task 1: Handle user interactions for setting and acknowledging reminders.
Task 2: Implement a calendar view to navigate through scheduled doses.

Day 5: State Management and Combine Framework
Task 1: Manage app state for scheduled reminders and use data binding to reflect changes instantly when a medication is taken or skipped.
Task 2: Use the Combine framework to handle changes in medication schedules and trigger notifications.

Day 6: Lists, Animation, and Adaptivity
Task 1: Utilize list views in SwiftUI to show daily medication schedules and animate updates as medications are marked off.
Task 2: Adapt the app UI to different screen sizes and implement gesture controls for snoozing reminders.

Day 7: Advanced SwiftUI Topics and Core Data Integration
Task 1: Design an adaptive user interface that supports accessibility features like dynamic text sizing and voice-over.
Task 2: Integrate the app with Core Data to persistently store medication data and schedule information.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.
3. Provide references for any external sources used.

Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day.

Module
Duration (Days

Sl. No.
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9

1.10
1.11
1.12
1.13
1.14
2.0
2.1
2.2
3.0
3.1
3.2
3.3
3.4
3.5

3.5
4.0
4.1
4.2
4.3
4.4
5.0
5.1
5.2
5.3
5.4
5.5
5.6
6.0

6.1
6.2
6.3
7.0
7.1
7.2
8.0
8.1
8.2
8.3
8.4
8.5
9.0

9.1
9.2
9.3
10.0
10.1
10.2
10.3
11.0
11.1
11.2
11.3
12.0
12.1

12.2
12.3
12.4
12.5
13.0
13.1
13.2
13.3
14.0
14.1
14.2
14.3
14.4
14.5

14.6
14.7
14.8
14.9
15.0
15.1
15.2
15.3
15.4
16.0
16.1
16.2
16.3
17.0
17.1
17.2
18.0

18.1
18.2
18.3
18.4
18.5
19.0
19.1
19.2
19.3
19.4
19.5
20.0
20.1
20.2
20.3
20.4
20.5
20.6
21.0
21.1
21.2
21.3
21.4

21.5
21.6
21.7
21.8
22.0
22.1
22.2
22.3
22.4
22.5

Appium based Testing
15.00

Topics
Software Testing Fundamentals
Introduction to Software Testing
Understanding Software Testing: Definitions and Importance
Different Types of Software Testing
The Role of a Software Tester
Software Testing Fundamentals
Overview of Software Testing
Definition and Importance of Software Testing in the SDLC
Goals and Principles of Software Testing
Levels of Software Testing
Introduction to Unit Testing, Integration Testing, System Testing, and Acceptance Testing
Understanding the Objectives, Target, and Environment of Each Level
Software Development Life Cycle (SDLC) and Testing Life Cycle (STLC)
Overview of SDLC Phases
Introduction to STLC and Its Phases
Relationship between SDLC and STLC
Software Testing Techniques
Static vs. Dynamic Testing Techniques
White-box, Black-box, and Grey-box Testing
Basics of Manual Testing
Test Planning and Design
Understanding Requirements
Test Case Development
Test Data Preparation
Test Execution and Bug Reporting
Executing Test Cases
Identifying and Logging Defects
Defect Life Cycle
Types of Testing
Functional Testing
Including Regression, Smoke, Sanity, and User Acceptance Testing
Non-Functional Testing
Including Performance, Load, Stress, Usability, and Security Testing
Test Planning
Creating Test Strategies and Test Plans
Resource and Environment Planning
Test Design
Writing Test Cases and Test Scripts

Designing Tests Based on Requirements and Specifications
Test Execution
Executing Test Cases and Recording Results
Managing Test Cycles
Defect Management
Defect Lifecycle Management
Reporting and Tracking Defects Using Tools
Introduction to Automation Testing
What is Automation Testing?
Benefits of Automation Testing Over Manual Testing
When to Choose Automation Testing
Introduction to Performance Testing
Understanding Performance Testing
Key Concepts: Load Testing, Stress Testing, and Capacity Testing
Tools for Performance Testing
Introduction to Security Testing
Importance of Security Testing
Common Security Testing Approaches
Basic Security Testing Tools and Techniques
Introduction to Appium
Appium Features
Cross-platform testing capabilities
Support for native, hybrid, and web application testing
Compatibility with different programming languages
Appium Internal Architecture
Client/Server Architecture
Overview of the Appium server, nodes, and sessions
Communication process between Appium client and server
Appium Installation Instructions for Windows and Mac
Download Java, Android Studio, and Node Software for Appium Setup
Version compatibility and selection criteria
Setting Environment Variables
For Windows
For Mac
Verification process to ensure correct setup
Configuring Android Virtual Device/Emulator
Creating and managing AVDs for different Android versions and screen sizes
Tips for optimizing performance of the emulator
Installing Appium Server
Methods of installation: NPM and Desktop App
Validating successful installation through command line and GUI
Setting Up Appium Maven Project with Java Client Dependencies
Creating a new Maven project in IDEs such as Eclipse or IntelliJ

Adding Appium Java client dependencies in the POM file
Running a sample test to ensure correct setup
Getting Started with Mobile Testing Using Appium Code
Understanding Desired Capabilities
Introduction to Desired Capabilities in Appium
Configuring Desired Capabilities for Android and iOS
Hands-on Exercise: Setting Up Your First Appium Project
First Program to Invoke an App
Writing the First Test Script
Launching an App using Appium Server
Debugging Common Issues When Invoking an App
Introduction to UIAutomatorViewer
What is UIAutomatorViewer?
Inspecting Elements in Android Apps
Generating Xpath, IDs, and other Selectors for Use in Scripts
Inspecting Objects in Android Apps
Deep Dive into Object Identification Strategies
Understanding the UI Hierarchy
Practical Session: Inspecting and Interacting with Various UI Elements
Android Native Apps Automation with Appium
Automating app UI Using Xpaths and Text Attributes
XPath Syntax and Best Practices
Using Text Attributes to Locate Elements
Appium APIs for UI Interaction
Overview of Appium's API for UI Interactions
Using IDs and classNames for Element Interaction
Advanced Interaction Techniques
Handling Multiple Similar Objects with Indexes
Strategies for Dealing with Lists and Grids
Using Indexes to Identify Objects
Using AndroidUIAutomator for Object Identification
Exploring AndroidUIAutomator's Advanced Features
Writing Custom Selectors Using AndroidUIAutomator Syntax
TouchActions in Appium
Understanding the TouchActions Interface
Implementing Swipe, Scroll, Drag-and-Drop
Gesture Automation
Gesture Automation -1: Basic TouchActions
Gesture Automation -2: Tapping and Long Press Events
Gesture Automation -3: Touch Actions Sequence Execution
Strategy for Automating Swiping Events
Advanced Gestures Automation with Appium
Demo: Swiping the Clock

Understanding the Swipe Gesture in Detail
Practical Session: Automating Time Setting on a Clock App
Challenges in Swiping Gestures and Overcoming Them
Gesture Automation -5: Automating Scrolling
The Mechanics of Scroll Gestures in Mobile Apps
Custom Scrolling Functions for Different Scenarios
Hands-on: Scrolling Through a Long List in a Social Media App
Gesture Automation -6: Drag and Drop on Native Apps
Understanding Drag and Drop with Appium
Strategies for Effective Drag and Drop Automation
Workshop: Building a Test for Reordering Items in a To-Do List
Real Android Device Automation with Appium
Configuring a Real Device for Appium Tests
Setting Up Devices for Testing with Developer Options
Establishing Connection Between the Device and the Test Environment
Verifying the Setup with a Basic Test
Native Apps Automation Exercises on Real Devices
Practical Differences When Testing on Real Devices Versus Emulators
Addressing Common Challenges such as Notifications and Calls
Automating a Series of Real Device Tests
Real-Time Ecommerce App Testing
Introduction to App Features and Test Cases to Automate
Overview of Common Ecommerce App Features
Identifying Key Test Scenarios and Cases for Ecommerce Apps
Planning End to End Test Flows
Form Details Filling and Toast Message Verification
Automating User Input on Forms
Validation of Form Submissions and Error Handling
Extracting and Asserting Toast Messages for Feedback
Product List Scrolling and Dynamic Selection
Strategies for Testing Infinite Scrolling Features
Dynamic Product Selection Based on Various Criteria
Automation Techniques for Asserting Selected Products
Checkout Page and Total Amount Validation
Automating the Checkout Process
Validating Calculations and Total Amounts
Ensuring Accuracy of Transactions through Automation
Code Optimization Techniques
Refactoring Code for Better Maintenance and Readability
Implementing Page Object Model for Ecommerce Scenarios
Utilizing Design Patterns for Scalable Test Scripts
Hybrid App Automation with Appium
Mobile Gestures Validation (Tap, Long Press)

Detailed Exploration of Tap and Long Press in Hybrid Contexts
Gesture Implementation Challenges on Hybrid Apps
Practical Exercises: Incorporating Gestures into Hybrid App Tests
Handling Web View Rendered Objects
Understanding the Nuances of Automating Web Views in Appium
Techniques for Switching Contexts Between Native and Web Views
Best Practices for Element Interaction within Web Views
Context Switch in Hybrid App Automation
Mastering the Context Switch for Hybrid Apps
Managing Multiple Contexts within a Single Test Flow
Code Along Session: Automating a Hybrid App with Context Switching
Mobile Chrome Browsers Automation with Appium
Setting Desired Capabilities for Mobile Browser
Configuring Appium for Mobile Browser Testing
Establishing Desired Capabilities for Chrome on Android Devices
Cross-Browser Testing Strategies
Inspecting Chrome Mobile Browser Locators
Tools and Techniques for Inspecting Web Elements in Chrome
Differences Between Desktop and Mobile Browser Inspections
Practicing Locator Strategies for Mobile Web
Scrolling and Assertions in Browser Tests
Implementing Automated Scrolling in Mobile Browsers
Assertions and Validations During Web Navigation
Case Study: Automated Testing of a Web Application in Chrome Mobile Browser
iOS Automation Using Appium on MAC
Installing Xcode and iPhone Simulator
Preparing the MAC Environment for iOS Testing
Installing and Configuring Xcode
Setting Up the iPhone Simulator for Testing
Defining iOS Desired Capabilities and App Invocation
Tailoring Desired Capabilities for iOS Devices
Launching an iOS App Through Appium
Troubleshooting Common Issues with iOS App Invocation
iOS Objects Inspection with Appium Inspector
Leveraging Appium Inspector for iOS Object Identification
Exploring iOS UI Components with Appium Inspector
Scripting Efficient iOS Tests with Accurate Element Locators
iOS Advanced Gesture Automation with Appium
iOS Scrolling Automation
Techniques for Scrolling Through Views in iOS Apps
Custom iOS Scroll Functions for Various Use Cases
Automating Vertical and Horizontal Scrolling
Automating Picker Wheels (Dropdowns)

Strategy for Handling Picker Wheels in iOS
Writing Scripts to Interact with Date Pickers and Custom Picker Wheels
Setting Date and Time in Calendar Apps
Automating Sliders with iOS Element Class
Approaches to Automate Slider Controls in iOS
Adjusting Volume and Brightness Settings in an App
iOS Gestures Strategy and App Download
Formulating Strategies to Automate Complex Gestures
Automating App Downloads from the App Store
Testing User Flow for Downloading and Installing New Apps
Long Press and Tapping Events with iOS Touch Action
Implementing Long Press and Tap Gestures
Adding Items to Favorites via Long Press Actions
Creating Multi-Touch Actions for Advanced User Interactions
iOS Real Device Automation Testing
Real Device Automation Software Installation
Installing and Setting Up Necessary Tools for Real Device Testing
Verifying Environment Setup with a Connectivity Test
UDID, XcodeOrgId, and Bundle Id Generation
Finding and Using the UDID for iOS Devices
Generating and Configuring XcodeOrgId and Bundle Id for Apps
Practical Steps to Set Up a Real Device for Automation
Running Tests on a Real Device
Executing Automated Tests on iOS Real Devices
Troubleshooting Common Real Device Testing Issues
Appium Framework - TestNG Basics
Installation and Setup in Eclipse
Installing and Configuring TestNG in Eclipse IDE
Setting Up an Appium Project with TestNG Integration
Running Basic Tests with Annotations
Understanding TestNG Annotations for Efficient Testing
Creating and Running Simple TestNG Tests
Exploring the TestNG XML File and Its Structure
TestNG Xml File Configuration
Configuring TestNG XML Files for Custom Test Suites
Organizing Tests with TestNG XML
Test Case Prioritization and Inclusion/Exclusion Mechanisms
Prioritizing Test Execution Order
Including and Excluding Tests from Execution
Managing Test Dependencies
Package Level Test Execution
Executing Tests at a Package Level for Structured Test Management
Utilizing Regex Patterns for Dynamic Test Selection

Deep Dive into TestNG Annotations
Advanced Use of TestNG Annotations for Complex Test Scenarios
Annotations for Pre and Post Test Activities
Group Functionality and Annotations' Helper Attributes
Organizing Tests into Groups for Targeted Test Runs
Using Helper Attributes to Enhance Test Cases
Parameterizing Tests with DataProvider Annotation
Implementing Data-Driven Testing with TestNG
Using DataProviders for Parameterized Tests
TestNG Listeners and Parallel Test Execution
Incorporating Listeners for Additional TestNG Functionality
Executing Tests in Parallel to Save Time
Maven Integration in Appium Framework
Maven Introduction and Configuration
Understanding the Role of Maven in Build Management
Configuring Maven for Appium Projects
Maven Project Creation and POM.xml Understanding
Creating a New Maven Project
In-depth Analysis of the POM.xml File and Its Components
Surefire Plugin and TestNG Integration
Using the Maven Surefire Plugin for Test Execution
Integrating TestNG with Maven for Better Test Management
Maven Profiling for Test Switching
Understanding Maven Profiles
Using Profiles to Manage Different Testing Environments and Configurations
Jenkins – Continuous Integration Tool
Importance of Jenkins in Test Frameworks
Introduction to Continuous Integration and Jenkins
The Role of Jenkins in Automating the Build and Test Process
Install and Configure Jenkins
Step-by-Step Guide to Installing Jenkins on Different Operating Systems
Configuring Jenkins for Appium Test Execution
Configuring Jenkins Settings and Workspace
Setting up Jenkins Job to Run Appium Tests
Managing Jenkins Workspace and Job Configurations
Pageobject Pattern and Page Factory
What is Page Object Model?
Understanding the Concept of Page Object Model (POM)
Benefits of Using POM in Test Automation
Page Factory Annotations FindBy
How to Use Page Factory Annotations for Element Locators
Writing Robust and Maintainable Test Code with Page Factory
Appium Hybrid Framework Design - Design Pattern

Maven Project Template Creation with Workspace Setup
Creating a Standard Maven Project Structure for Appium Tests
Setting Up the Development Workspace
Configuring Dependencies of the Project in Pom.xml
Creating Driver Base class for Framework
Designing a Driver Base Class for Appium to Handle Driver Initializations
Global Properties Initialization
Implementing Global Properties Management for Multi-Environment Support
Page Objects Implementation in Framework
Appium Hybrid Framework Design - Building Utilities
Make Tests Objects Independent to Drive from Object Classes
Developing Independent Test Objects for Reusability and Modularity
Programmatically Managing Appium Server
Writing Utility Methods to Manage Appium Server Lifecycle Within Tests
Building Common Utilities for Appium Server Invoke Code
Creating a Utility Library to Support Common Appium Functions
Starting Emulator on Fly Through Code with User Inputs
Automating the Emulator Start-Up Process Based on Test Requirements
Fixing Framework Bug by Debugging Code
Techniques for Debugging and Resolving Issues in the Automation Framework
Reporting and CI/CD Integration for the Appium Framework – Part 3
Capturing Screenshot on Test Failures Using TestNG Listeners
Implementing Listeners to Capture Screenshots on Test Failures for Debugging
Generating Client Html Reports for Test Execution
Customizing TestNG HTML Reports
Parameterizing Test Data from External Sources Using Data Provider
Leveraging External Data Sources Like Excel, CSV, or Databases for Data-Driven Testing
Convert Project to Maven Compatible by Updating pom.xml
Ensuring the Project is Fully Mavenized for Dependency Management and Build Processing
Sending Global Properties from Maven Commands into Tests
Dynamically Injecting Configuration Properties into the Test Suite via Maven Commands
Integrating Framework Repo into Jenkins CI Tool
Finalizing the CI/CD Pipeline by Integrating the Appium Test Framework with Jenkins for Conti
Database Connection to Selenium/Appium Test Cases
Steps to Connect Database to Selenium Test Cases
Integrating JDBC for Database Connectivity in Tests
Executing Database Queries from Test Code
MySQL Download Instructions
Setting Up MySQL for Use with Selenium/Appium Tests
MySQL Server Connection Procedure
Establishing a Connection to MySQL Server from Test Code
Creating Database in MySQL Server
Basic SQL Commands for Creating and Managing Databases

JDBC ODBC Connection Overview
Understanding JDBC-ODBC Bridge for Database Connectivity
Integration of Database with JDBC API
Utilizing JDBC API for Robust Database Operations in Test Automation
Steps to Connect Database Info to Selenium - 1
Steps to Connect Database Info to Selenium - 2
Defect Tracking Tool
Defect Management tool- Bugzilla
Introduction to Bugzilla for Defect Tracking
Setting Up and Configuring Bugzilla
Bugzilla Features- Logging a Bug
How to Log and Manage Bugs in Bugzilla
Bugzilla Features -Search Mechanism for Tracking Bugs
Utilizing Bugzilla's Search Features to Track and Manage Defects
Common Practices in Bugzilla Tool
Best Practices for Using Bugzilla in Test Management
Bugzilla Preference Feature
Customizing Bugzilla for Individual or Team Preferences

Back		
Duration (Days)		
1.00		

1

1		
1		

1		
1		

1		
1		

1		
1		
1		

1		

1		
0.75		
0.75		

0.5		

Module
Duration (Days)

Capstone Project
5

Capstone

Assignment 1

Development Scenario: Third Eye Automotive Surveillance

Day 1: Automotive Security Introduction and Environment Setup, Camera Integration and Motion Detection
Task 1: Learn about automotive security systems, focusing on camera-based surveillance for moving and parked vehicles.
Task 2: Set up an Android development environment and create a project that includes permissions for camera access and background processing.
Task 3: Integrate the camera API to capture live video feeds and still images from within an Android application.
Task 4: Implement basic motion detection algorithms that trigger photo capture when movement is detected around the vehicle.

Day 2: Vibration Detection and Event-Driven Capture
Task 1: Develop a system to detect vibrations using the vehicle's built-in sensors or external hardware connected to an Android device.
Task 2: Set up an event-driven mechanism that initiates the camera to take photos every 2 seconds when vibrations are detected while parked.
Task 3: Process and compress images captured by the camera to reduce storage space without compromising quality.
Task 4: Efficiently store the captured images and videos locally with a timestamp and geo-tagging information.

Day 3: Power Management and Background Services
Task 1: Design a power management system that minimizes battery usage when the vehicle is parked.
Task 2: Implement a background service that runs the surveillance system only when the vehicle is locked and the engine is off.

Day 4: Data Transmission and User Interface
Task 1: Create a method for transmitting alerts and images to the vehicle owner’s smartphone or designated device.
Task 2: Develop a user interface for the owner to interact with the Third Eye system, including viewing the captured media and receiving alerts.

Day 5: Testing, Security, and Compliance
Task 1: Perform rigorous testing in various scenarios, including different lighting and weather conditions, to ensure the system’s reliability.
Task 2: Ensure that all data capture and transmission are secure and comply with privacy laws and automotive regulations.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing

tool.
3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day.

Scenario

Assignment 2

Development Scenario: Panic Button Safety Application

Day 1: Introduction to Android System Services and Environment Setup/ Hardware Interaction and User Interface
Task 1: Understand Android's system-level services, focusing on broadcast receivers and services that respond to hardware button presses.
Task 2: Set up an Android Studio project with permissions for SMS, location, and camera access.
Task 3: Implement a broadcast receiver to detect long-press actions on the power and volume buttons.
Task 4: Create a user interface for the confirmation alert dialog that appears after the panic mode is triggered.

Day 2: Location Services, SMS Integration and Real-Time Data Transmission
Task 1: Integrate location services to fetch the user's current location.
Task 2: Develop the functionality to send an SMS to predefined contacts with the user's location if the confirmation alert is not dismissed within 5 seconds.
Task 3: Establish a service that starts upon panic mode activation and sends location updates via SMS every 5 seconds.
Task 4: Integrate the camera API to capture photos periodically and send MMS to registered contacts.

Day 3: Testing and Reliability
Task 1: Conduct thorough testing to ensure the hardware button press detection is reliable and doesn't produce false positives.
Task 2: Test the application's performance in various scenarios, including low battery conditions and poor network connectivity.

Day 4: Security and Privacy Compliance
Task 1: Implement security measures to protect the transmission of sensitive data (like location and photos).
Task 2: Ensure the application complies with privacy laws regarding the sharing of location and multimedia data.

Day 5: Deployment and User Education
Task 1: Prepare the application for deployment, including setting up a beta testing phase.
Task 2: Develop user documentation and quick-start guides that educate users on the safe and effective use of the panic button feature.

Submission Guidelines:
1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.
2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.

3. Provide references for any external sources used.
Submit your work in a single PDF document by end of Module.
4. You must submit your code on gitlab by the end of next day

