Index

JEE FULL STACK 2.0 WITH REACT JS - (8 WEEKS)

JEE Full Stack 2.0 with React JS variant provides exposure to the entire spectrum of Java technologies starting from Core Java to Spring. It focuses on Web Application development using React JS and Spring Technology. The following table lists the course structure.

Sr. No.	Course	Duration	Immersive approach Remarks
	Discover (Induction)	--	
1	Soft Skills Foundation – Part 1	1	
2	Core Java 8 + Database & PostGreSQL with DevOps (Git, Sonarqube, Gradle, Jenkin)	12.5	**Project kick off-Individual project use cases has to be implemented**
3	**Core Java 8 Test**	**0.5**	Coding and MCQ Test
4	Soft Skills Foundation – Part 2	1	
5	JPA with Hibernate with PostgreSQL	**2**	
6	Spring 5.0 (Core + MVC + REST + Data JPA + Data REST)	**6**	Wireframe diagram to be created for case study
7	Soft Skills Foundation – Part 3	1	
8	Sprint 1 + JPA and Spring MCQ Test	5	Sprint 1- Backend implementation using Spring REST and Spring Data JPA
8	**Sprint 2 Evaluation**	**1**	
10	Soft Skills Foundation – Part 4	1	
11	HTML 5, CSS 3 with bootstrap 4, JavaScript ES6	4	
12	React JS (Jasmine Node as Server)	6	
13	**Sprint 2 +React JS + MCQ Test**	**4**	Sprint 2 - Front end implementation using React
14	Sprint 2 Evaluation	1	
15	L1 Preparation	**1**	
16	L1 Test	**1**	
	Total Training Duration	**48**	

Execution:

- **Week 1 – Participants to complete Agile coursera course to understand Agile methodology before Project Kick off to understand and use daily scrum meeting, Project Backlog, Sprint Backlog, Sprint review.**
- **Week 1 Project Kick off – Expectation setting by BU Mentor**
    - **Sprint Planning , Group formed , Case study shared . (BU Support)Declarations and**
- **Week 2 - Requirement review ,Understanding and Design artifacts , Use case , class , sequence diagram to prepared**
- **Week 4 - Artifacts Reviewed and Functional requirement design through interfaces , Test Cases.**
- **Sprint 1 implementation with code reviews of L&D and BU trainer**
    - **Test case reviews**
    - **Code reviews**
    - **Performance monitoring during the sprint implementation and sharing the feedback**
    - **Sprint - 1 Evaluation 30mins/participant**

- **Sprint 2 implementation with code reviews of L&D and BU trainer**
    - **Test case reviews**
    - **Code reviews**
    - **Performance monitoring during the sprint implementation and sharing the feedback**
    - **Sprint - 2 Evaluation 30min/participant**

Core Java 8

Program Duration: 10.5 days

Contents:

- **Declarations and Access Control**
    - Identifiers & JavaBeans
    - Legal Identifiers
    - Sun's Java Code Conventions
    - JavaBeans Standards
    - Declare Classes
    - Source File Declaration Rules
    - Class Declarations and Modifiers
    - Concrete Subclass
    - Declaring an Interface
    - Declaring Interface Constants
    - Declare Class Members
    - Access Modifiers
    - Nonaccess Member Modifiers
    - Constructor Declarations
    - Variable Declarations
    - Declaring Enums

- **Object Orientation**

- Encapsulation
- Inheritance, Is-A, Has-A
- Polymorphism
- Overridden Methods
- Overloaded Methods
- Reference Variable Casting
- Implementing an Interface
- Legal Return Types
- Return Type Declarations
- Returning a Value
- Constructors and Instantiation
- Default Constructor
- Overloaded Constructors
- Statics
- Static Variables and Methods
- Coupling and Cohesion

- **Assignments**
  - Stack and Heap—Quick Review
  - Literals, Assignments, and Variables
  - Literal Values for All Primitive Types
  - Assignment Operators
  - Casting Primitives
  - Using a Variable or Array Element That Is Uninitialized and Unassigned
  - Local (Stack, Automatic) Primitives and Objects
  - Passing Variables into Methods
  - Passing Object Reference Variables
  - Does Java Use Pass-By-Value Semantics?
  - Passing Primitive Variables
  - Array Declaration, Construction, and Initialization
  - Declaring an Array
  - Constructing an Array
  - Initializing an Array
  - Initialization Blocks
  - Using Wrapper Classes and Boxing
  - An Overview of the Wrapper Classes
  - Creating Wrapper Objects
  - Using Wrapper Conversion Utilities
  - Autoboxing
  - Overloading
  - Garbage Collection
  - Overview of Memory Management and Garbage Collection
  - Overview of Java's Garbage Collector
  - Writing Code That Explicitly Makes Objects Eligible for Garbage Collection

- **Operators**
  - Java Operators
  - Assignment Operators

- Relational Operators
- instanceof Comparison
- Arithmetic Operators
- Conditional Operator
- Logical Operators

- **Flow Control, Exceptions**
  - if and switch Statements
  - if-else Branching
  - switch Statements
  - Loops and Iterators
  - Using while Loops
  - Using do Loops
  - Using for Loops
  - Using break and continue
  - Unlabeled Statements
  - Labeled Statements
  - Handling Exceptions
  - Catching an Exception Using try and catch
  - Using finally
  - Propagating Uncaught Exceptions
  - Defining Exceptions
  - Exception Hierarchy
  - Handling an Entire Class Hierarchy of Exceptions
  - Exception Matching
  - Exception Declaration and the Public Interface
  - Rethrowing the Same Exception
  - Common Exceptions and Errors

- **Gradle Fundamentals**
  - Introduction
  - Folder Structure
  - Install and Setup Gradle on Windows
  - Dependencies in Build Scripts
  - Gradle Wrapper
  - Lifecycle Tasks: The Base Plug In
  - Using Project Info and the check command
  - Creating Variables and external properties
  - Creating a Build Scan
  - Dependencies

- **TDD with Junit 5**
  - Types of Tests
  - Why Unit Tests Are Important
  - What's JUnit?
  - JUnit 5 Architecture

- IDEs and Build Tool Support
- Setting up JUnit with Maven
- Lifecycle Methods
- Test Hierarchies
- Assertions
- Disabling Tests
- Assumptions
- Test Interfaces and Default Methods
- Repeating Tests
- Dynamic Tests
- Parameterized Tests
- Argument Sources
- Argument Conversion
- What Is TDD?
- History of TDD
- Why Practice TDD?
- Types of Testing
- Testing Frameworks and Tools
- Testing Concepts
- Insights from Testing
- Mocking Concepts
- Mockito Overview
- Mockito Demo
- Creating Mock Instances
- Stubbing Method Calls

- **Strings, I/O, Formatting, and Parsing**
  - String, StringBuilder, and StringBuffer
  - The String Class
  - Important Facts About Strings and Memory
  - Important Methods in the String Class
  - The StringBuffer and StringBuilder Classes
  - Important Methods in the StringBuffer and StringBuilder Classes
  - File Navigation and I/O
  - Types of Streams
  - The Byte-stream I/O hierarchy
  - Character Stream Hierarchy
  - RandomAccessFile class
  - The java.io.Console Class
  - Serialization
  - Dates, Numbers, and Currency
  - Working with Dates, Numbers, and Currencies
  - Parsing, Tokenizing, and Formatting
  - Locating Data via Pattern Matching
  - Tokenizing

- **Generics and Collections**
  - Overriding hashCode() and equals()

- Overriding equals()
- Overriding hashCode()
- Collections
- So What Do You Do with a Collection?
- List Interface
- Set Interface
- Map Interface
- Queue Interface
- Using the Collections Framework
- ArrayList Basics
- Autoboxing with Collections
- Sorting Collections and Arrays
- Navigating (Searching) TreeSets and TreeMaps
- Other Navigation Methods
- Backed Collections
- Generic Types
- Generics and Legacy Code
- Mixing Generic and Non-generic Collections
- Polymorphism and Generics

- **Threads**
  - Defining, Instantiating, and Starting Threads
  - Defining a Thread
  - Instantiating a Thread
  - Starting a Thread
  - Thread States and Transitions
  - Thread States
  - Preventing Thread Execution
  - Sleeping
  - Thread Priorities and yield( )
  - Synchronizing Code
  - Synchronization and Locks
  - Thread Deadlock
  - Thread Interaction
  - Using notifyAll( ) When Many Threads May Be Waiting

- **Concurrent Patterns in Java**
  - Introducing Executors, What Is Wrong with the Runnable Pattern?
  - Defining the Executor Pattern: A New Pattern to Launch Threads
  - Defining the Executor Service Pattern, a First Simple Example
  - Comparing the Runnable and the Executor Service Patterns
  - Understanding the Waiting Queue of the Executor Service
  - Wrapping-up the Executor Service Pattern
  - From Runnable to Callable: What Is Wrong with Runnables?
  - Defining a New Model for Tasks That Return Objects
  - Introducing the Callable Interface to Model Tasks
  - Introducing the Future Object to Transmit Objects Between Threads

- Wrapping-up Callables and Futures, Handling Exceptions

- **Concurrent Collections**
  - Implementing Concurrency at the API Level
  - Hierarchy of Collection and Map, Concurrent Interfaces
  - What Does It Mean for an Interface to Be Concurrent?
  - Why You Should Avoid Vectors and Stacks
  - Understanding Copy On Write Arrays
  - Introducing Queue and Deque, and Their Implementations
  - Understanding How Queue Works in a Concurrent Environment
  - Adding Elements to a Queue That Is Full: How Can It Fail?
  - Understanding Error Handling in Queue and Deque
  - Introducing Concurrent Maps and Their Implementations
  - Atomic Operations Defined by the ConcurrentMap Interface
  - Understanding Concurrency for a HashMap
  - Understanding the Structure of the ConcurrentHashMap from Java 7
  - Introducing the Java 8 ConcurrentHashMap and Its Parallel Methods
  - Parallel Search on a Java 8 ConcurrentHashMap
  - Parallel Map / Reduce on a Java 8 ConcurrentHashMap
  - Parallel ForEach on a Java 8 ConcurrentHashMap
  - Creating a Concurrent Set on a Java 8 ConcurrentHashMap
  - Introducing Skip Lists to Implement ConcurrentMap
  - Understanding How Linked Lists Can Be Improved by Skip Lists
  - How to Make a Skip List Concurrent Without Synchronization

- **Lambda Expressions**
  - Introduction
  - Writing Lambda Expressions
  - Functional Interfaces
  - Types of Functional Interfaces
  - Method reference
- **Stream API**
  - Introduction
  - Stream API with Collections
  - Stream Operations

Introduction to Design Pattern

Self learning with online links and explanation by Trainer with Demos

- Creational Design Pattern
  - Factory Pattern

        - Singleton Pattern
        - Prototype Pattern
    - Structural Design Pattern
        - Decorator Pattern
        - Facade Pattern
        - Proxy Pattern
    - Behavioral Design Pattern
        - Chain Of Responsibility Pattern
        - Iterator Pattern
        - Observer Pattern
        - State Pattern

    - Presentation Layer Design Pattern
        - Intercepting Filter Pattern
        - Front Controller Pattern
        - View Helper Pattern
    - Business Layer Design Pattern
        - Business Delegate Pattern
        - Session Facade Pattern
        - Transfer Object Pattern
    - Integration Layer Design Pattern
        - Data Access Object Pattern

DevOps/ CI CD concepts (GitHub/Nexus, CI Jenkins, Sonar)

Contents:

- Introduction to DevOps :
    - What is DevOps
    - Evolution of DevOps
    - Agile Methodology
    - Why DevOps
    - Agile vs DevOps
    - DevOps Principles
    - DevOps Lifecycle
    - DevOps Tools
    - Benefits of DevOps
    - Continuous Integration and Delivery pipeline
    - Use-case walkthrough

- GitHub
    - What is DevOps
    - Introduction to Git
    - Version control
    - Repositories and Branches
    - Working Locally with GIT
    - Working Remotely with GIT

- Jenkins
  - Introduction to CI
  - Jenkins Introduction
  - Creating Job in Jenkins
  - Adding plugin in Jenkins
  - Creating Job with Maven & Git
- Jenkins With TDD(Junit testing)
  - Integration of jUnit testing with Jenkins
- Sonar

Database Using PostgreSQL
Duration : 2 days
Contents:

- **Introduction**
  - The Relational Model
  - What is PostgreSQL?
  - PostgreSQL – Data Types
  - Arrays Functions and Operators
- **Understanding Basic** PostgreSQL **Syntax**
  - The Relational Model
  - Basic SQL Commands - SELECT
  - Basic SQL Commands - INSERT
  - Basic SQL Commands - UPDATE
  - Basic SQL Commands – DELETE
- **Querying Data with the SELECT Statement**
  - Wildcards (%, _)
  - The SELECT List
  - SELECT List Wildcard (*)
  - The FROM Clause
  - How to Constrain the Result Set
  - DISTINCT and NOT DISTINCT
- **Arrays Functions and Operators**
  - array_append
  - array_cat
  - array_lower
  - array_to_string
  - array_agg
  - every,Count,sum,avg
  - Array Operators
- **Filtering Results with the Where Clause**

- WHERE Clause
- Boolean Operators
- The AND Keyword
- The OR Keyword
- Other Boolean Operators BETWEEN, LIKE, IN, IS, IS NOT

- **Shaping Results with ORDER BY and GROUP BY**
  - ORDER BY
  - Set Functions
  - Set Function And Qualifiers
  - GROUP BY
  - HAVING clause

- **Matching Different Data Tables with JOINs**
  - Table Aliases
  - CROSS JOIN
  - INNER JOIN
  - OUTER JOINs
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
  - SELF JOIN
  - Natural Join

- **Creating Database Tables**
  - CREATE DATABASE
  - CREATE TABLE
  - NULL Values
  - PRIMARY KEY
  - CONSTRAINT
  - ALTER TABLE
  - DROP TABLE
- **PostgreSQL Transactions**
  - BEGIN, COMMIT, ROLLBACK

- **PostgreSQL Constraints**
  - CHECK, UNIQUE, NOT NULL

    - **Introduction to JDBC**
      - **Connection, Statement, PreparedStatement, ResultSet**

JPA Using PostgreSQL

Program Duration: 2 days

Contents:

- **Introduction**

- Introduction & overview of data persistence
- Overview of ORM tools
- Understanding JPA
- JPA Specifications

- **Entities**
  - Requirements for Entity Classes
  - Persistent Fields and Properties in Entity Classes
  - Persistent Fields
  - Persistent Properties
  - Using Collections in Entity Fields and Properties
  - Validating Persistent Fields and Properties
  - Primary Keys in Entities
- **Managing Entities**
  - The EntityManager Interface
  - Container-Managed Entity Managers
  - Application-Managed Entity Managers
  - Finding Entities Using the EntityManager
  - Managing an Entity Instance's Lifecycle
  - Persisting Entity Instances
  - Removing Entity Instances
  - Synchronizing Entity Data to the Database
  - Persistence Units
- **Querying Entities**
  - Java Persistence query language (JPQL)
  - Criteria API
- **Entity Relationships**
  - Direction in Entity Relationships
  - Bidirectional Relationships
  - Unidirectional Relationships
  - Queries and Relationship Direction
  - Cascade Operations and Relationships

Spring 5.0

Program Duration: 6 days

Contents:

1. **Spring Core**

Spring Core Introduction / Overview

- Shortcomings of Java EE and the Need for Loose Coupling
- Managing Beans, The Spring Container, Inversion of Control
- The Factory Pattern
- Configuration Metadata - XML, @Component, Auto-Detecting Beans
- Dependencies and Dependency Injection (DI) with the BeanFactory
- Setter Injection

Spring Container

- The Spring Managed Bean Lifecycle
- Autowiring Dependencies

Dependency Injection

- Using the Application Context
- Constructor Injection
- Factory Methods
- Crucial Namespaces ‘p’ and ’c’
- Configuring Collections

Metadata / Configuration

- Annotation Configuration @Autowired, @Required, @Resource
- @Component, Component Scans. Component Filters
- Life Cycle Annotations
- Java Configuration, @Configuration, XML free configuration
- The Annotation Config Application Context

2. Spring MVC

Introduction / Developing Web applications with Spring MVC

- The WebApplicationContext and the ContextLoaderListener
- Model View Controller
- Front Controller Pattern
- DispatcherServlet Configuration
- Controllers, RequestMapping
- Working with Forms
- Getting at the Request, @RequestParam, @RequestHeader
- ModelAndView

Advanced Techniques

- Spring form tags and Model Binding, @ModelAttribute

Spring Controllers

- Using @ResponseBody
- JSON and XML data exchange

RESTful Web Services

- Core REST concepts
- REST support in Spring 5.x
- Use Spring MVC to create RESTful Web services
- REST specific Annotations in Spring
- Working with RestTemplate
- URITemplates, @PathVariable, @RequestParam
- JSON and XML data exchange
- @RequestMapping

3. Spring Boot

SPRING BOOT Introduction

- Spring Boot starters, CLI, Gradle plugin
- Application class
- @SpringBootApplication
- Dependency injection, component scans, Configuration
- Externalize your configuration using application.properties
- Context Root and Management ports
- Logging

Using Spring Boot

- Build Systems, Structuring Your Code, Configuration, Spring Beans and Dependency Injection, and more.

Spring Boot Essentials

- Application Development, Configuration, Embedded Servers, Data Access, and many more
- Common application properties
- Auto-configuration classes
- Spring Boot Dependencies

4. Spring Data JPA

- Spring Data JPA Intro & Overview
- Core Concepts, @RepositoryRestResource
- Defining Query methods
- Query Creation
- Using JPA Named Queries
- Defining Repository Interfaces
- Creating Repository instances
- JPA Repositories
- Persisting Entities
- Transactions

5. Spring Data REST

- Introduction & Overview
- Adding Spring Data REST to a Spring Boot Project
- Configuring Spring Data REST
- Repository resources, Default Status Codes, Http methods
- Spring Data REST Associations
- Define Query methods

6. Introduction to Spring Security with Demo
7. Introduction to Spring Microservices with Demo

HTML 5, CSS 3 with Bootstrap, Javascript ES6

Program Duration: 4 days

Commented [SS1]: Basic HTML tags like UL, LI, HREF, images should be prerequisits and can be excluded from here.

Contents:

HTML 5:

- HTML Basics
  - Understand the structure of an HTML page.
  - New Semantic Elements in HTML 5
  - Learn to apply physical/logical character effects.
  - Learn to manage document spacing.
- Tables
  - Understand the structure of an HTML table.
  - Learn to control table format like cell spanning, cell spacing, border
- List
  - Numbered List
  - Bulleted List
- Working with Links
  - Understand the working of hyperlinks in web pages.
  - Learn to create hyperlinks in web pages.
  - Add hyperlinks to list items and table contents.
- Image Handling
  - Understand the role of images in web pages
  - Learn to add images to web pages
  - Learn to use images as hyperlinks
- Frames
  - Understand the need for frames in web pages.
  - Learn to create and work with frames.
- HTML Forms for User Input
  - Understand the role of forms in web pages
  - Understand various HTML elements used in forms.
  - Single line text field
  - Text area
  - Check box
  - Radio buttons
  - Password fields
  - Pull-down menus
  - File selector dialog box
- New Form Elements
  - Understand the new HTML form elements such as date, number, range, email, search and datalist
  - Understand audio, video, article tags

CSS 3

- **Introduction to Cascading Style Sheets 3.0**
  - What CSS can do
  - CSS Syntax

Commented [SS2]: The saved time from reducstion of HTML section can be used here to learn modular CSS using SASS. Most of the projects use LESS or SCSS for CSS pre compilers and it will good to have knowledge of it.

- Types of CSS
- **Working with Text and Fonts**
  - Text Formatting
  - Text Effects
  - Fonts
- **CSS Selectors**
  - Type Selector
  - Universal Selector
  - ID Selector
- Class selector
- **Colors and Borders**
  - Background
  - Multiple Background
  - Colors RGB and RGBA
  - HSL and HSLA
  - Borders
  - Rounded Corners
  - Applying Shadows in border

- Implementing CSS3 in the "Real World"
  - Modernizr
  - HTML5 Shims
  - SASS, and Other CSS Preprocessors
  - CSS Grid Systems
  - CSS Frameworks

BootStrap

- **Introduction to Bootstrap**
  - Introduction
  - Getting Started with Bootstrap
- **Bootstrap Basics**
  - Bootstrap grid system
  - Bootstrap Basic Components
- **Bootstrap Components**
  - Page Header
  - Breadcrumb
  - Button Groups
  - Dropdown
  - Nav & Navbars
- **JavaScript Essentials**
- **ES6**
  - Var, Let and Const keyword
  - Arrow functions, default arguments
  - Template Strings, String methods
  -
  - Object de-structuring

- Create,apply,prototype,bind method
- Spread and Rest operator
- Typescript Fundamentals
- Types & type assertions, Creating custom object types, function types
- Typescript OOPS - Classes, Interfaces, Constructor, etc
- Decorator & Spread Operator
- Difference == & ===
- Asynchronous Programming in ES6
- Promise Constructor
- Promise with Chain
- Promise Race
-

React JS

Program Duration: 6 days

Contents:

- **Getting started with Node.js**
  - JavaScript Essentials
  - How JavaScript works
  - Event loop
  - Stack, Heap and Queue
  - Node.js Fundamentals
  - Introduction to Node.js
  - Why Node.js?
  - Traditional Programming Limitations

- **React Introduction**
  - Overview of frameworks, libraries for client side Web applications
  - React introduction,
  - Understanding "what" and "why" React
  - React Component Demonstration using codepen
  - Environment Setup for React Application
  - Understanding NPM commands
  - Using VS Code
  - VS Code extensions for ES6, React(formatting and checkstyles)
  - Helloworld app in React
- **React Essential Features and Syntax**
  - React App Project Directory Structure
  - Overview of Webpack, Babel

Commented [SS3]: We should cover extensions which used for code formatting and checkstyles. Should be max 30 min.

- React Component Basic
- Create React Component
- Understanding JSX
- Limitations of JSX
- Working with Components and Reusing Components

- **React Components, Props and State**
  - Understanding and using Props and State
  - Handling Events with methods
  - Manipulating the State
  - Two way data-binding
  - Functional (Stateless) VS Class (Stateful) Components
  - Parent – Child Communication
  - Dynamically rendering contents
  - Showing Lists, List and keys
- **Styling Components**
  - CSS Styling
  - Scoping Styles using Inline Styles
  - Limitations of inline styes
  - Inline Styles with Radium
- **Google Material UI**
  - Installing Material UI
  - Material UI AppBar
  - Material UI's Toolbar
  - Custom React NavBar CSS
  - Material UI Buttons
  - Using Material UI - Rendering a Button
  - Material UI Card
  - Material UI Checkbox
  - Material UI Grid Component
  - Material UI IconButton
  - Material UI Paper Component
  - Style Material UI Components with my own CSS
  - UI Templates for Business
  - Typography Usage
- **Debugging React Apps**
  - Understanding React Error Messages
  - Handling Logical Errors,
  - Debugging React apps using google developer tools and React DevTool
  - Understanding Error Boundaries
- **React Component life cycle**
  - Updating life cycle hooks
  - PureComponents
  - React's DOM Updating Strategy
  - Returning adjacent elements
  - Fragments
- **React Component in Details**
  - Higher Order Components

- Passing unknown Props
- Validating Props
- Using References
- React Context API
- Updated LifeCycle hooks (16.3)
- Best practices for React Projects
- Demo apps

- **HTTP Requests/Ajax Calls**
  - HTTP Requests in React
  - Introduction of Axios package
  - HTTP GET Request, fetching & transforming data
  - HTTP POST, DELETE, UPDATE
  - Handing Errors
  - Adding/Removing Interceptors
  - Creating/Using Axios intances
  - Redux
  - React Thunk
  - Difference between Thunk & other
  - React hooks
  - Application Using React & Redux

- **React Routing**
  - Routing and SPAs
  - Setting Up the Router Package
  - react-router vs react-router-dom
  - Preparing the Project For Routing
  - Switching Between Pages, Routing-Related Props
  - The "withRouter" HOC & Route Props
  - Passing & extracting route/query parameters
  - Using Switch to Load a Single Route
  - Navigating Programmatically
- **React Forms and Form Validation**
  - Creating a Custom Dynamic Input Component
  - Setting Up a JS Config for the Form
  - Dynamically Create Inputs based on JS Config
  - Adding a Dropdown Component
  - Handling User Input
  - Handling Form Submission
  - Adding Custom Form Validation
  - Fixing a Common Validation
  - Adding Validation Feedback
  - Showing Error Messages
  - Handling Overall Form Validity
- Deploying React App to the Web
- Testing React apps with Jasmine & implementining JEST

Commented [SS4]: DEMO app should be somewhat realistic. We can refere to Master class content to use same concept.