Enterprise Programming Project - Trade Message Routing

Welcome to the banking world. There's a bunch of trade junkies with way too much money (not their own) always looking for faster and faster ways to make their trades and analyze the market.

You want that money, so you told them you'd build them a system that would handle sending their trade messages electronically to other financial entities and even *transform* those messages so that their recipients can get the data in a format they prefer.

"What do we build it with?"

The first question should actually be "What am I building?" but this is Enterprise Software, so first we'll slap down the cold hard reality that you will be working in Java.

As that sinks in, we'll ease off a bit and let you can use modern Java. More on the tech stack later...

"What am I building, exactly?"

A system that gets a Trade Message from one client's system, to N number of recipient clients and possibly returns a response. Along the way, the message may need to be *transformed*. ***The exact details of who it's coming from, where it's going to, and what it looks like before and after is not set in stone***.

○ Job 01 - Basic Routing and Transformation

Your first two clients, we'll call them "Bank of Charles" (BOC) and "Zippo Stock Exchange" (ZSE) want something tame sounding. BOC has a trade message in their **proprietary XML format**, and they want to **send it to your system via FTP**, so you can **transform it into ZSE's "standard trade XML" format** and send it to them via FTP, and vice versa.

> In reality, you likely wouldn't send anything to an Exchange via FTP. It would likely go over something called the "SWIFT Network" but we are not banks operating at the billion-dollar mark, so we can't access that.

Oh, and they want you to **keep a record of all these messages going in and out** for auditing purposes. And your manager wants you to make sure your project has good test coverage. He gave no further elaborations.

"... Are we building this from scratch?"

Not unless you're a masochist. The clients, of course, want this done yesterday (the possibility that you can pull tailor made software out of thin air is an ever tangible one for business types). If we wanna hit deadlines, or at least not blow so far past them that we lose the contract, we need to build off existing technologies.

So what will you use?

As a freebie, we'll set the core framework of our system to *Apache Camel*. It's an "Integration Framework" based in Java that comes with a plethora of ready-to-use components to do almost anything you need to send and receive data over the various methods available. That means network protocols like HTTP, FTP, etc.

over the internet, or communicating with a DB on the local network, or even picking up and dropping files off the host machine's hard drive.

It's best to think of Camel as a LEGO box of Enterprise Software functionality. It even now has its own runtime handler, so it can run standalone if desired.

Also, we won't go rooting around for build systems and default to using Maven. Maven will handle the process of managing dependencies, building all of *your* code, and bundling it together into a shippable artifact. In exchange, it determines some of the structure of the project on the filesystem.

What I suggest first for any additional technology research, is first look through Camel's component library to see what technologies it can support out of the box. (Like, say, the FTP Component)

Other Software that might be helpful:

- Camel's own Kameleon Tool can generate the project boilerplate for you
- JetBrain's IntelliJ IDE is my favorite, and they have a free Community edition
- JUnit 5 is a testing framework for Java that integrates well with Maven projects (can be included with Kameleon)
- XML Parsing libraries (I'll leave researching which one or multiple to use as part of the lesson; they are all supported in Camel):
    - Jackson XML
    - DOM
    - JAXB
    - SAX
    - StAX
- SQL Databases
    - MySQL for a *standalone* database
    - Apache Derby for an *embedded* database
    - H2 for an *in-memory* database
    - ***H2 and maybe Derby do not need to be manually downloaded. They should be able to be managed by Maven as dependencies.***
    - Why three options? Here's an article with a brief explanation of the three types .
- FTP
    - For the server this article explains how to set one up using Windows's built-in software.
    - For the client, WinSCP or Filezilla are common.
        - The FTP client would be useful for you to pretend to be a business client pushing files to the server for the routing application to pick up.

... Hey, aren't we missing something?

Missing what?