1. In your opinion, what are the main advantages of using Spring in a webapp context ?

The Spring Web framework is used to build webapps using an MVC design pattern. This design pattern works because most web applications are structured similarly and generally all require a model, view and controller (at least) in order to meet basic requirements.

Another benefit of the Spring Web framework is that it encourages a loosely coupled design where each layer or component can be structured and tested in isolation.

A ServiceLocator alternative could be used instead but this would require a lot of extra code to do what Spring has already done and it probably wouldn’t be as clean as Spring if you were to do it yourself unless you spent a considerable amount of time on it.

Spring also offers a number of other classes that can be used in a webapp context such as an ORM abstraction for popular ORM’s like Hibernate

Sometimes the webapp lifecycle can be repetitive, Spring also has AOP capabilities so that annotations can imply that selective code should be run at certain stages of an application’s execution

Aside from all this, there is also a number of other Spring libraries such as Boot, Cloud, Data, Security, Web Services and more.

2. When would you use automatic testing ?

In my experience it’s best to ensure that projects first have unit, integration and system tests before moving onto automated tests. Automated testing is generally done by automated testers who have experience in BDD libraries such as Cucumber and Jbehave. In the context of webapps, however, Selenium is also a commonly used tool in automated testing. One important aspect of automated testing is reporting where feedback from tests can be provided back to technical and non-technical members to improve team collaboration. Generally, a developer will not push their work on a continuous integration tool such as Jenkins unless their project-level tests are passing on their local environment. Once an application is pushed to another environment, it’s important to have System properties in place to avoid a lot of environment related issues. However, these environment related issues generally get picked up quite quickly and the developer is able to rollback and try again at a later stage. In an environment where continuous delivery is used, changes can be pushed into the pipeline as soon as they’re ready, without having to wait for the current sprint to finish first. In this case, the same testing processes will take place but a product owner will benefit from being able to see smaller changes more frequently.

3. You just received the root access of your project's freshly installed linux box. Describe the main steps you'll take to run the war on it

Using Spring Boot, you can leverage off of an embedded tomcat instance. However, assuming that you need your tomcat to be hosted separately, the first thing you’ll need to do is install it. However, it’s important that you first perform an apt-get update and purge openjdk and install the official jdk as this is the jdk that is most widely used commercially. Once you have a jvm, you can then download a tomcat tar from the apache website and extract it somewhere convenient to you. Once extracted, you would go into the bin directory and execute the sh file to get it running. If you plan on using the manager page, you can change the tomcat-users.xml file to add your own manager-gui user which will then allow you access to deploy and undeploy war files once running. Most default settings on tomcat are good to go by default but it’s also a good idea to know how much memory your WAR file(s) will be using so that you can provision the container to have sufficient resources upon deployment. Assuming you only have one server, you won’t have to worry too much about clustering but Tomcat does, however, allow you to update the server.xml file to specify clustering options and also leverages off Apache HTTP server for load balancing where session affinity can be customised.

4. Once deployed, your users start whining about slow pages, timeouts, failed uploads, etc. As a developer, how would you handle this ?

In most cases non-production environments will pick this up. Once in production, your defences are down and unfortunately there’s no way around an upset user which is why it’s important to avoid upsetting users as much as possible by improving tests and continuous delivery. Considering that the users are all complaining about performance related issues in this scenario, there are a number of factors to consider on what could be causing this issue. i.e.

-Have I provisioned enough memory for the application?

-Have I introduced multi-threading in a way that could be causing a memory leak?

-Am I using any old technology that just isn’t going to do the trick?

-Could I have created a memory leak in any other way?

-Is this an issue I should’ve picked up profiling the application before releasing it or an issue that automated tests should’ve profiled before pushing into production?

-Is the server currently just over-burdened? If so, has it been on the edge for a long period of time and now is going over what it’s capable of doing?

-How is my production environment currently architected? Is it able to provision to the number of users I’m serving?

For the short-term I would use my continuous integration environment to rollback to a previously working version (assuming that it has no database impact, otherwise I might have to backup and rollback the database as well and then figure out a way to get any new data back into place at a later stage). For memory leaks, profiling and multi-threading I would spend time going through the code, fixing any issues I find, perhaps pair with a colleague to provide a second opinion if need be. If the issue is an architectural or infrastructural one then I’d review what we have and our usage reports to determine if there’s anything that can be improved such as placing static data on a CDN such as CloudFront or Akamai. If it’s an API endpoint issue then I’d look into whether it’s worthwhile upgrading the EC2 or other AWS specs or introducing more nodes into a clustered environment where they can work in parallel. In terms of updating old technology for performance, it’s not that common but from my experience I know that some developers have found some success when switching between Spring JDBC, Views, Stored Procs, Hibernate, etc depending on the infrastructure