Q. 1: If the code-base grew massively, would it affect the system design and if yes, how?

Yes, the massive code base would affect the system design drastically, due to which the coupling in our code base will increase and cohesion will decrease. In order to make code base less coupled we have to maintain modularization and break the services into different standalone services, so to maintain low coupling and high cohesion, which are the standard goals of high reliability and maintainability in software development.

In order to manage the massive code base, system design should be scalable enough to cater any future requirement. The technology stack often affect the scalability , so in order to achieve a balanced scalable design technology stack should be efficient enough because it is pivotal in the software development to follow design principles.

In my social networking app, I have chosen monolithic design over micro service architecture because it is a prototype application. In order to make this app production ready for millions of end users I would recommend micro service architecture because of it is reliable and easy to maintain.

Q. 2: Describe in your own words the testing strategy you would impose if the quality of a service similar to the one you designed but with more endpoints, more complex logic and many users was your responsibility?

Below are the testing strategy that I would implement in production ready apps:

To follow above procedure, I would ensure that my CI/CD pipeline is up to date and all pull/merge requests have been reviewed and checked. All dependencies are up to date and new database migration is done. Once the pipeline is ready, we can start the service.

The above pattern/guidelines are just a baseline requirement to enable test cases to be run properly.

It is worth to mention that testing in monolithic versus micro service architecture is altogether different. Application based on monolithic architecture are much easy to test then latter. The major issues in micro service architecture is availability, fragmented structure and holistic testing.

Few points to mention:

Keep up to date documentation in GIT as markdown and maintain API docs in swagger or any other similar platform.

Below are useful testing tools:

Swagger, VCR, Hoverfly, etc.

……. *End …….*