Backend Testing

Unit Testing

Unit Testing is performed using JUnit +MockK, with an aim of covering all functions in all important classes (this excludes data classes, enum classes, or other trivial object classes etc.). For each function, all clear pathways are tested; typically this is a happy path and a sad/error path. It is worth noting that since Kotlin is strongly typed and enforces certain type rules (although also allows type inference), many unhappy pathways are not possible by design, such as passing Integer where String is required (file would not compile in this case). Given this, many test cases are assumed impossible, or are only testable in integration testing - see below. GitHub actions is used to ensure that all tests are passing before any branch is merged to main; in general, the team's strategy is to always cover a new class/function with a unit test in the same PR to ensure poor code does not enter the repository (although there was some catching up to do in early weeks before a testing strategy was devised).

Integration Testing

Integration testing revolves around the endpoints exposed by the backend. The team mostly uses Postman for integration testing, with the shared team account. All endpoints that are used in the system are regularly tested to ensure the entire system works from rest controller through to database.

A typical testing regime is to first run the patient and appointment file upload endpoints to populate a database, before running further tests on this real data, as retrieved from Genie.

Environments

The team uses a number of environments to test the backend. Each member has a MySQL database running locally, allowing them to build the backend and test it on real data. This should always be done before raising a PR that includes a new or modified endpoint to ensure no broken endpoints exist on main.

The team also has its own server environment which always contains an instance of the backend running for testing. An updated backend instance should be built and uploaded to the shared testing environment as often as possible, with longer than 2-3 days being dispreffered if regular changes are occurring. This regular update allows the team to continually test their changes in conjunction with other pushes from team members, as well as exposing endpoints to the front end team for full integration testing.

The final environment is the real production environment used by the client, which is an EC2 instance running on the clinic's server. This environment is not fully set up as of writing, but it is aimed to be built into the production pipeline by the team before Sprint 3.