**SmartRPA Assignment**

1. Aspects of the assignment that are a requirement
   1. Java (Spring)
   2. Web-based
   3. REST API
   4. Frontend:
      1. Image with quote
      2. Quote hall of fame
2. Decisions made before starting the exercise
   1. Spring Initializr
      1. https://start.spring.io/
   2. Create a Controller-Service-Model architecture
   3. RestTemplate vs WebClient (high availability setup)
      1. **WebClient**: complicated, but non-blocking -> fewer system resources. Reactive programming!!
      2. RestTemplate: simple, but blocking and synchronous
   4. Security, RBAC?
      1. Only later
   5. Get a working API “MVP”, then improve on it
3. Decision made during development
   1. Define a Quote model based on the Simpsons response
   2. Start writing tests early
   3. Let’s store API Key properly for ninjas quotes.
   4. Let’s improve on exception handling
   5. Use swagger for API docs
   6. Backend is functional enough, let’s start working on the UI (React + TS)
      1. <https://create-react-app.dev/docs/adding-typescript/>
         1. create-react-app is deprecated. https://react.dev/blog/2025/02/14/sunsetting-create-react-app
      2. <https://vite.dev/guide/>
   7. Let’s create cards to display the simpsons and ninjas data
   8. Hybrid approach to reactive programming (BE vs FE)
   9. **MVP is done**, lets continue with filtered quotes
   10. Filtering works, lets write tests for it.
       1. 401 Unauthorized due to spring security -> Update WebFluxTest annotation to include the SecurityConfig class
       2. Tests could be improved
   11. Before worrying about css, lets finish all tasks
   12. How do I implement a hall-of-fame feature?
       1. DB storage options:
          1. **Relational DB (Hibernate) (In-memory vs Postgres)**
          2. NoSQL DB
          3. In-memory (Redis?, ConcurrentHashMap?)
       2. Storing votes:
          1. Simple counter
          2. User-based (maybe later)
          3. IP-based?
          4. Session-based?
4. Hurdles
   1. Working with Mono and Flux
   2. <https://intellij-support.jetbrains.com/hc/en-us/community/posts/23064675521682-Lombok-not-workin-with-Intellij>
   3. Not really a hurdle, but interesting: MockBean is deprecated, use MockitoBean instead
   4. How exactly should the service be tested?
      1. Mocking every step of a Mono
      2. Mockito’s deep stubs
      3. **StepVerifier** vs assertEquals
   5. CORS error
      1. **Netty** vs Tomcat
      2. Finally introduced Spring Security
   6. App loads twice
   7. How to swap between quotes and hall of fame?
      1. Landing page, then navigate
      2. **Land on the random quotes** and have the chance to swap to HoF
   8. Styling
   9. How to toggle between 5-10-20 quotes?
      1. Get 20, only display the required amount
      2. **Refetch each time**
   10. How should I allow multiple users to use the application?
       1. Keycloak? -> too much overhead
       2. Built-in Auth? -> May not be perfect, but easier
   11. In-memory Hibernate is not a good solution, lets introduce Postgres
5. Happy moments
   1. API gives back a Simpsons quote as a String: [{"quote":"Nothing you say can upset us. We're the MTV generation.","character":"Bart Simpson","image":"https://cdn.glitch.com/3c3ffadc-3406-4440-bb95-d40ec8fcde72%2FBartSimpson.png?1497567511638","characterDirection":"Right"}]
   2. Introduced a QuoteModel, and the code still works
   3. Introduced a NinjasModel, the application can give back both simpsons and ninjas quotes
   4. Tests are in place, and work for both Simpson and ninjas quotes
   5. Swagger has successfully been integrated into the application
   6. Vite app has been created, introduced types that match the models, client receives the quotes from the server
   7. Quotes are properly displayed
   8. New quotes can be fetched
   9. Filtering is enabled
   10. Hibernate has successfully been integrated, quotes are saved, voting is enabled (BE only)
   11. Voting works on the FE as well, Hall of fame has been introduced
   12. Keycloak has been introduced
   13. Voting is fixed
   14. Postgres and Keycloak have been dockerized, Keycloak realm, client, role and users are set up automatically
6. Task statuses:
   1. **The backend must be written in Java ✔**
   2. **The application must be web-based (i.e. not a command line application) ✔**
   3. **The application must expose its own REST API that has at least one endpoint that can fetch a quote from one of the aggregated APIs at random ✔**
   4. Some other feature suggestions:
      1. fetch a more specific quote from some input that the endpoint accepts ✔
      2. a voting system so users can vote for quotes that they like, resulting in a kind of “hall-of-fame” of the best 5-10-20 quotes ✔
      3. returning the quotes in the hall-of-fame as a list sorted so that the highest ranked quotes are first ✔
      4. contain a permission-based system allowing various levels of access – for instance the open part could be what is described above and a part requiring authentication could allow adding new quote APIs to the aggregator**✔**
   5. Consider how to support simultaneous use – both on the same app instance and in a high availability setup where several application instances might run against the same database
   6. Consider and possibly create tests of the application ✔
   7. **The application must have some (perhaps very rudimentary) frontend that shows at least a bit more than the raw response from the API ✔**
   8. Frontend suggestions:
      1. Display an image if one is associated with a quote ✔
      2. Show the quote hall of fame✔