Technical Steps

Design: Das Programm „Sports Exercise Battle“ benutzt C#, PostgreSQL, eine Datenbank (DBeaver & Docker) zusammen mit einem HTTP-Server ohne helper Framework. Die Funktionalitäten wurden aufgeteilt in BLL, DAL und eine API-Layer.

Failures: Zu den Problemen die ich während der Implementation hatte, zählt auf jeden Fall die Arbeit mit der Datenbank. Um spezifischer zu sein – bestimmte Sachen abzuspeichern, die später gelöscht werden sollten, z.B. laufende Turniere im Falle eines Server-crash. Hierfür wurde eine Funktion implementiert, die wenn das Programm neu gestartet wird, alle Turniere auf den Wert NULL setzt und löscht. Somit kann man garantieren, dass wenn das Programm rebootet / neu gestartet wird, es keine laufenden Turniere mehr gibt. Ein weiteres Problem war auch die Implementation eines Timers. Vor allem für mehrere Turniere habe ich mir die Frage gestellt, wie man am besten eine dynamische Lösung entwickelt. Hierfür wurde dann eine Liste von Timern erstellt, in der jeder Timer einem eigenen Turnier zugeordnet wird. Das mit Abstand größte Problem welches ich hatte, war die StartTime zu setzen. Anfangs wurde diese hardcoded aber im Nachhinein wurde wurde es in einem Weg implementiert, indem jeder Timer zusammen mit einem TournamentSchema in einem Concurrent Dictionary gespeichert Wird, welches die Startzeit erhält. Durch dieses Schema besteht die Möglichkeit, jedem Turnier seinen eigenen Timer zuzuordnen, was dazu führt, dass mehrere Turniere gleichzeitig ohne Hardcoding der Startzeiten existieren können.

Lessons Learned: Durch dieses Projekt konnte ich meine Skills, wenn es um arbeiten mit Datenbanken geht, deutlich verbessern. Zusätzlich wurde verstanden, wie Payloads aus Curl-Scripts verarbeitet werden und wie man eigene Curl-Script Commands zu seinem Vorteil nutzen kann. In meinem Fall wurden z.B. mehrere Curl-Script Commands eigenständig hinzugefügt um implementierte Funktionalitäten zu testen.

Unit Test Design

Es wurden die wichtigsten Funktionalitäten innerhalb des Programms ausgewählt um Unit Tests zu implementieren. Dazu zählen: Score, Tournament und User.

Scores:

- InsertUserStats_CallsDaoInsertUserStats:
  In diesem Unit Test wird verifiziert, dass wenn „ScoreManager.InsertUserStats“ mit einem spezifischen Token gecalled wird, genau mit demselben token nur einmal „IScoreDao.InsertUserStats“ gecalled wird. Dieser Test bestätigt, dass die Daten wie erhofft an das DAO weitergegeben werden.

- GetScoreBoard_ReturnsListOfUserStats:
  In diesem Unit Test wird die „ScoreManager.GetScoreboard“ Methode getestet. Es wird geprüft, ob genau das zurückgegeben wird, was das DAO vorgibt. Der Test mockt das DAO um ein prädefiniertes UserStatsSchema zu returnen und verifiziert, dass „ScoreManager.GetScoreboard“ diese Liste genauso wiedergibt. Dieser Test bestätigt, dass eine Liste von dem DAL abruft und korrekt wiedergibt.

- GetSpecificUserStats_WhenUserExists_ReturnsUserStats:
  Hier wird die Methode „GetSpecificUserStats“ getestet. Es wird überprüft, ob diese Methode so wie sie sollte den Token an das DAO weitergibt und die erhofften Statistiken returned.

- AddElo_CallsDaoAddEloWithCorrectAmount:
  Dieser Test bestätigt, dass die richtigen Parameter (Elo Punkte, AuthToken) von der AddElo Methode an das DAO weitergegeben werden um die Elo Punkte zu updaten.

Tournament:

- GetHistory_WithNoHistory_ThrowsEmptyHistoryException:
  Der Test überprüft das Verhalten der „GetHistory“ Methode, wenn keine History für einen User vorhanden ist. Die RetrieveHistory Methode wurde nämlich so konfiguriert, dass sie eine leere Liste zurückgibt, wenn sie für den Benutzer „nonexistentuser“ aufgerufen wird. Deshalb wird von diesem Unit Test eine Fehlermeldung gethrowt.

- GetHistory_WithHistory_ReturnsHistory:
  In diesem Unit Test wird die „RetrieveHistory“ Methode so konfiguriert, dass sie eine Liste von HistorySchemas zurückgibt, wenn sie für den „testuser“ aufgerufen wird. Es sollte sozusagen eine History für diesen Benutzer simulieren. Danach wird überprüft, ob die Einträge auch tatsächlich existieren so wie sie sollten.

User:

- RegisterWithValidCredentialsDoesNotThrow:
  Überprüft, dass kein Error gethrowt wird, wenn ein neuer Benutzer mit gültigen Anmeldedaten registriert wird. (RegisterUser)

- RegisterWithExistingUserTHrowsDuplicateUserException:
  Hier wird sichergestellt, dass eine „DuplicateUserException“ gethrowt, wenn ein bereits existierender User erneut registriert wird.

- LoginWithValidCredentialsDoesNotThrowException:
  Der Test überprüft, ob ein Fehler auftritt, wenn sich ein User mit validen Login-Daten einloggt. Es wird ebenfalls sichergestellt, dass die „UserLogin“ Methode des DAO genau einmal aufgerufen wird.

- LoginWithInvalidCredentialsThrowsUserNotFoundException:
  Dieser Unit Test stellt sicher, dass eine „UserNotFoundException“ gethrowt wird, wenn sich ein Benutzer mit ungültigen Daten anmelden will. Auch hier wird überprüft, dass „UserLogin“ genau einmal aufgerufen wird.

Time Spent

Das Projekt wurde am 9. März angefangen und es wurde in 1-Wochen-Abständen (Aufgrund anderer Fächer) daran weitergearbeitet. Insgesamt beläuft sich die verbrachte Zeit auf ungefähre 16 Stunden, aufgrund der Probleme die in Kapitel 1 genannt wurden.

GIT-Repository:

https://github.com/philippsawa/SWEN_KOMP