Лабораторная 1.

Цель: получить практический опыт применения акторов.

Необходимо реализовать поисковой агрегатор, который по запросу пользователя собирает top 5 ответов через API известных поисковиков и выдает их пользователю. Например, делает запрос в Google, Яндекс, Bing и возвращает 15 ответов (должно быть ясно, какой ответ от какой поисковой системы). Реальное API можно не использовать, а реализовать StubServer, который будет отдавать результаты в удобном формате (json, xml, protobuf и тд).

Архитектура приложения:

- На каждый запрос создается master-actor, который будет собирать результаты от поисковиков
- Master-actor для каждого поисковика создает child-actor, которому посылает исходный "поисковый запрос"
- Master-actor устанавливает себе receive timeout, сколько времени он будет ждать ответы от child-actors
- Child-actor делает запрос в соответствующий поисковый сервис и отправляет его результат в master-actor
- Master-actor при получении каждого ответа сохраняет их у себя, если получил все 3 ответа или прошло время таймаута, то отправляет собранный агрегированный результат на дальнейшую обработку
- Master-actor должен умирать после того, как вернул агрегированный результат

Указания:

- В stub-server необходимо реализовать возможность залипания, чтобы можно было проверить сценарий, когда master-actor не дождался ответов от всех поисковиков
- Примеры из лекции: https://github.com/akirakozov/software-design/tree/master/java/akka

Баллы:

- 20 баллов
- необходимо добавить тесты (без тестов -5 баллов)

Лабораторная 2.

Цель: получить практический опыт реализации "реактивных" приложений.

Необходимо реализовать веб-сервис для просмотра каталога товаров. В сервисе можно регистрировать новых пользователей и добавлять товары. Пользователи при регистрации указывают, в какой валюте они хотят видеть товары (доллары, евро, рубли).

Указания:

- Веб-сервис должен быть полностью "реактивным" (все взаимодействие должно быть асинхронным и не блокирующим);

- В запросах можно не проверять авторизацию пользователей, а просто указывать id пользователя;
- Данные можно хранить в mongo и использовать "реактивный" mongo driver; в качестве http-сервиса можно использовать rxnetty-http;
- Можно использовать и другие реактивные библиотеки и фреймворки;
- Пример из лекции: https://github.com/akirakozov/software-design/tree/master/java/rxjava

Баллы:

- 20 баллов

Указание:

- задание можно сдавать через e-mail, в заголовке письма указать “[SD-TASK]”
- в письме указать ссылку на ваш код на github.com

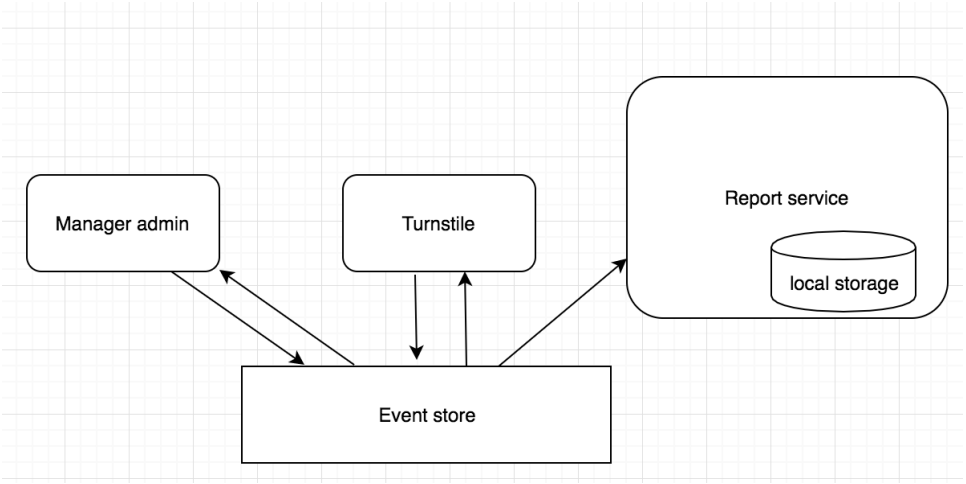
Лабораторная 3.

Цель: получить практический опыт реализации event-sourcing систем и применения принципов CQRS.

Необходимо реализовать информационную систему для фитнес-центра. Клиенты фитнес-центра могут приобретать и продлевать абонементы. При входе и выходе из фитнес-центра, клиент обязан приложить абонемент для прохода. Система позволяет строить различные отчеты на базе статистики собранной в ходе работы сервиса.

Система состоит из трех частей:

1. Веб сервис для менеджера, позволяет
- Просматривать информацию о абонементах (по номеру)
- Выдавать и продлевать абонементы
2. Сервис отчетов, позволяет
- просмотреть статистику посещение по дням
- считать среднюю частоту и длительность посещений
3. Модуль входа/выхода
- пускает клиента, если у него действующий абонемент
- сохраняет в базе время входа/выхода клиента



Указания:

- Система должна быть построена на базе event-sourcing подхода
- Команды и запросы должны строго разделяться (CQRS-принцип)
- Сами события должны храниться в персистентном хранилище.
- Сервис для менеджера и модуль входа/выхода не имеют отдельной базы, и при запросе вытаскивают все события об абонементе из хранилища событий, агрегирую их на лету.
- Сервис отчетов, наоборот, при старте вычисляет статистику и сохраняет ее в памяти, обновляя новыми событиями (но можно и хранить в отдельном персистентном хранилище).

Ссылки:

- http://ookami86.github.io/event-sourcing-in-practice/

Баллы:

- 20 баллов

необходимо добавить тесты (без тестов -5 баллов)

Лабораторная 4.

Цель: получить практический опыт реализации интеграционных тестов с использованием docker и testcontainers.

Необходимо реализовать информационную систему для торговли на бирже. Для простоты, будем считать, что биржа просто хранит некоторый объем акций для каждой компании, у которых может динамически меняться цена. Любой пользователь может купить/продать необходимое ему число акций на бирже по текущей цене.

Система состоит из двух частей:

1. Эмулятор биржи, позволяет:

- Добавлять новые компании и их акции
- Узнавать текущую цену акций и их количество на бирже
- Покупать акции компаний по текущей цене
- Динамически менять курс акций

Механизм обновления курса может быть любой: можно ходить в реальные сервисы бирж, если найдете открытое api и обновлять; можно написать какой-то рандомайзер, который будет в некотором окне менять курс акций; можно сделать просто админку, через которую стоимость будет меняться вручную.

2. Личный кабинет пользователя, позволяет:

- Заводить новых пользователей (заходить можно без авторизации, просто по id)
- Добавлять денежные средства на счет пользователя
- Просматривать акции пользователя, их количество и текущую стоимость
- Просмотреть, сколько сейчас суммарно у пользователя средств, если считать все акции по текущей стоимости
- Покупать/продавать акции на бирже

Указания:

- Должна быть возможность упаковывать эмулятор биржи в docker-контейнер
- Для личного кабинета необходимо написать интеграционные с биржей тесты, которые будут использовать фреймворк testcontainers (https://www.testcontainers.org/) или его аналоги. Интеграционные тесты будут запускать эмулятор биржи в docker и ходить в него
- задание можно сдавать через e-mail, в заголовке письма указать “[SD-TASK]”
- в письме указать ссылку на ваш код на github.com

Пример:

- https://github.com/akirakozov/software-design/tree/master/java/test-containers-example - пример maven конфига, который при сборке добавляет в локальный docker registry java приложение.

Для общего развития, про уровни тестирования и цели интеграционного тестирования:

- https://martinfowler.com/articles/microservice-testing/
- https://martinfowler.com/articles/practical-test-pyramid.html