

Task 2. Fault tolerance

В рамках второй лабораторной работы необходимо модифицировать систему таким образом, чтобы обеспечить гарантированную обработку запроса пользователя (если доступен менеджер) т.е. обеспечить отказоустойчивость системы в целом.

Основные требования:

1. Обеспечить сохранность данных при отказе работы менеджера

- a. Для этого необходимо обеспечить хранение данных об обрабатываемых запросах в базе данных
- b. Также необходимо организовать взаимодействие воркеров с менеджером через очередь RabbitMQ
 - i. Для этого достаточно настроить очередь с direct exchange-ем¹
 - ii. Если менеджер недоступен, то сообщения должны сохраняться в очереди до момента возобновления его работы
- c. RabbitMQ также необходимо разместить в окружении docker-compose²

2. Обеспечить частичную отказоустойчивость базы данных

- a. База данных также должна быть отказоустойчивой, для этого требуется реализовать простое реплицирование³ для нереляционной базы MongoDB
- ~~b. Достаточно будет схемы с одной primary и одной secondary нодой⁴~~
Минимально рабочая схема одна primary нода, две secondary
- c. Менеджер должен отвечать клиенту, что задача принята в работу только после того, как она была успешно сохранена в базу данных и отреплицирована

3. Обеспечить сохранность данных при отказе работы воркера(-ов)

- a. В docker-compose необходимо разместить, как минимум, 2 воркера
- b. Организовать взаимодействие менеджера с воркерами через очередь RabbitMQ (вторая, отдельная очередь), аналогично настроить direct exchange
- c. В случае, если любой из воркеров при работе над задачей "сломался" и не отдал ответ, то задача должна быть переотправлена другому воркеру, для этого необходимо корректно настроить механизм acknowledgement-ов⁵
- d. Если на момент создания задач нет доступных воркеров, то сообщения должны дожидаться их появления в очереди, а затем отправлены на исполнение

4. Обеспечить сохранность данных при отказе работы очереди

¹ Гайд по RabbitMQ Exchanges

<https://habr.com/ru/post/489086/>

² Деплой RabbitMQ в docker-compose

<https://www.section.io/engineering-education/dockerize-a-rabbitmq-instance/>

³ Дока по реплицированию MongoDB

<https://www.mongodb.com/docs/manual/replication/>

⁴ Гайд по настройке реплицирования MongoDB в окружении docker-compose

<https://flowygo.com/en/blog/mongodb-and-docker-how-to-create-and-configure-a-replica-set/>

⁵ RabbitMQ Consumer Acknowledgements and Publisher Confirms

<https://www.rabbitmq.com/confirms.html>

- a. Если менеджер не может отправить задачи в очередь, то он должен сохранить их у себя в базе данных до момента восстановления доступности очереди, после чего снова отправить накопившиеся задачи
- b. Очередь не должна терять сообщения при рестарте (или падении из-за ошибки), для этого все сообщения должны быть персистентными⁶ (это регулируется при их отправке)

Кейсы, которые будут проверяться:

- 1. стоп сервиса менеджера в docker-compose
 - a. полученные ранее ответы от воркеров должны быть сохранены в базу и не должны потеряться
 - b. не дошедшие до менеджера ответы на задачи не должны потеряться, менеджер должен подобрать их при рестарте
- 2. стоп primary ноды реплик-сета MongoDB в docker-compose
 - a. primary нода должна измениться, в система продолжать работу в штатном режиме
- 3. стоп RabbitMQ в docker-compose
 - a. все необработанные, на момент выключения очереди, сообщения после рестарта не должны потеряться
- 4. стоп воркера во время обработки задачи
 - a. сообщение должно быть переотправлено другому воркеру, задача не должна быть потеряна

Примечания.

- 1. Для отправки и получения сообщений в формате xml в очередь лучше всего использовать `MarshallingMessageConverter` + `Jaxb2Marshaller` в связке с `AmqpTemplate`(отправка) и `@RabbitListener` (получение)
- 2. При рестарте RabbitMQ сбрасывает статус отправленных сообщений (персистентных) из `unacked` в `ready`. Поэтому, например, допускается повторная обработка одной задачи двумя воркерами, что нужно учесть в логике менеджера.

⁶ RabbitMQ Persistence Configuration
<https://www.rabbitmq.com/persistence-conf.html>