Практическая часть

Очень трудоемкое задание оказалось. Зато получил прекрасную возможность поизучать новый spring. Spring, конечно, добавил достаточно технических проблем. Не все я решил хорошо.

В нескольких местах отошел от текста задания:

Подозреваю, что в mysql полно своих особенностей. Их знание может пригодиться, чтобы выжать из mysql максимум. Я с mysql почти не работал и сделал бы так - использовал InnoDB и общие для всех баз оптимизации. Но на доступном мне сервере нет mysql и я там не админ. Поэтому будет embedded H2. Все равно нечего демонстрировать.

Сделал отдельные формы логина и регистрации потому, что хотел покрутить spring security и валидацию форм. С валидацией не получилось красиво реализовать ошибки полей для mustache - сделал костыль.

Версткой особо никогда не занимался - взял дизайн с игры mpets.mobi. Надо было выбрать другую игру - дизайн не соответствует тематике.

Известные баги:

Подсчет статистики выполнения запросов к бд для формы логина почему-то не работает. Spring security, наверное, специфично делает редиректы. Для формы регистрации статистика работает нормально.
Время работы фреймворка spring учитывается в статистике не полностью. (Реализовано при помощи Interceptor)

Предположительные баги:

Подсчет статистики реализован кривовато. Также, база в памяти не дает полезной статистики кроме количества запросов.
Возможно, неправильно работаю с транзакциями хибернейта. Надеюсь дефолтные настройки работают хорошо.

GIT https://github.com/repinpv/game-task
URL http://java-dev.tabatoune.com:8888

Теоретические вопросы

Задача 1

Условия:

Есть игра, в которую приходит 10 000 игроков и бьют мощного босса.
У игрока и босса есть два параметра – урон и жизни.
Босс выбирает случайного игрока и бьет его 1 раз в секунду пока у игрока не кончатся жизни, далее переключается на следующего.
Раз в минуту босс наносит небольшой урон всем игрокам.
Раз в две минуты босс наносит большой урон ста самым сильным игрокам (по параметру урон).
Игрок не чаще раза в секунду может наносить урон боссу.
Если по истечении 10 минут босс не убит, битва завершается.

Вопросы:

1. Как организовать хранение сущностей игрока и босса? Как обеспечить их взаимодействие?
2. Как корректно учитывать урон от игроков, чтобы он терялся?
3. Как сделать нанесение урона по всем игрокам? По ста самым сильным?
4. Какие дополнительные данные требуется хранить?

Решение:

Для хранения характеристик игроков в бд:

```
class PlayerCombatStats {
private int userId;
private int hp;
private int damage;
}
```

Для хранения характеристик босса в бд:

```
class BossCombatStats {
private int bossId;
```

```
private int hp;
private int singleAttackDamage;
private long singleAttackPeriod;
private int massAttackDamage;
private long massAttackPeriod;
private int massAttackForStrongerEnemiesDamage;
private long massAttackForStrongerEnemiesPeriod;
}
```

Бои с боссами я думаю повторяются периодически.
Для хранения в бд:

```
class RepeatedBossRaid {
private int id;
private int bossId;
private long beginTime;
private long endTime;
private long period;
private long announceLength;
private long length;
private long showResultLength;
}
```

На основе этого класса вычисляется текущий рейд на босса.

```
class CurrentBossRaid {
  private RepeatedBossRaid bossRaid;
  private long beginAnnounceTime;
  private long beginTime;
  private long endTime;
  private long showUntilTime;
}
```

Он хранится в сервисе рейдов в оперативной памяти ява-машины.

Было бы удобно для юзеров, чтобы 10000 юзеров приходили не заранее, а могли присоединиться к рейду в любое время до завершения рейда. Если всех юзеров собирать заранее до начала рейда, то очень просто можно рассчитать время смерти каждого и сразу сохранить в память. Далее рассматривается вариант с присоединением в любой момент до завершения рейда.

Объект этого класса хранит текущее состояние босса.

```
class BossState {
  private AtomicInteger hp;

  public void takeAttack(int damage) {
```

```
    Int prevHp = hp.getAndAdd(-damage);
    If (prevHp <= damage) {
      // process win
    }
  }
}
```
Тут уже нужно работать с потоками и отслеживать обнуление хп.

Каждый удар игроков нужно будет хранить в специальной базе. Они пригодятся для анализа и показа лога игрокам.

С ударами босса сложнее.

Создаем объект с ленивыми вычислениями времени следующей атаки босса:
```
class BossMassAttack {
  private long nextMassAttackTime;
  private int inflictedDamage;
  private long nextMassAttackForTopTime;
}
```
Он пригодится для вычисления времени смерти вновь присоединившихся игроков.

Состояние юзеров храним в мапах:
```
class PlayerState {
  private long attackCooldownTime;
  private int hpLevel; // increased hp for user joined after boss attacks
}

Map<Integer, PlayerState> alivePlayerStates = new ConcurrentMap<>();
```

В момент присоединения создается состояние игрока в памяти:

```
Int hpLevel = bossMassAttack.getInflictedDamage() + playerCombatStats.getHp();
playerStates.put(userId, new PlayerState(hpLevel));
```

Теперь вычисляем 100 сильнейших по дамаге игроков. Проблема, что топ постоянно меняется, поскольку приходят новенькие игроки и старые погибают. Если бы было известно, что дамага коррелирует с хп, это можно было бы использовать. А так придется ранжировать всех участников. Ещё одна проблема, что скорее всего будет много игроков с одинаковым дамагом и нужно добавить ещё одно правило для сортировки. Например, кто первый пришел, тот первый получает урон. Если игроки с одинаковым уроном каждый раз перемешиваются в топе, то игрокам будет непонятно, почему раз ударили, а второй раз нет. Ну и босс будет размазывать дамагу по игрокам. В итоге, стоит использовать принцип первый пришел - первый получил. При присоединении новых игроков, некоторых

старых тоже могут перестать бить. Но это заметно игроку по счетчику участников. Есть одно серьёзное упрощение задачи - дамага игроков не меняется во время рейда.

Используем базовые классы.
TreeMap<Integer (damage), LinkedList<Integer (userId)>>
alivePlayerSortedByDamageAndJoinTime;

При присоединении игрока, по дамаге игрока берем список и добавляем в конец.

При обработке массового удара сильнейших бежим реверсным итератором по мапе. Потом прямым итератором по каждому списку. Удаляем уже мертвых из списка. Отнимаем первым 100м хп удара и удаляем погибших.

Теперь последняя атака босса - одного игрока раз в секунду. Тут все просто. Выбираем одного игрока из alivePlayerStates и отнимаем ему дамаг раз в секунды.

Разберемся с потоками.

Есть потоки обработки запросов юзеров - присоединения к рейду, удар босса и обновления страницы. Запросов много и очень желательно, чтобы они ничего не блокировали. Если параллельно будут идти обработки массового удара босса, то можно их не ждать. Можно считать, что успел нанести удар.

Значит вид классов такой:

```
class PlayerState {
  private AtomicLong attackCooldownTime;
  private AtomicInteger hpLevel; // increased hp for user joined after boss attacks
}

class BossMassAttack {
  private long nextMassAttackTime; // используется в потоке обработки массового удара
  private AtomicInteger inflictedDamage;
}

Int hp = playerState.getHpLevel() - bossMassAttack.getInflictedDamage();
```

При присоединении нового игрока, добавляем его в alivePlayerStates и записываем информацию в конкурентный накопитель для топов.

Потоки очень легковесны. Тем более раз в секунду, минуту и две минуты. Но почему бы не бить боссом в одном потоке.

Запускаем задачу с периодом в 1 секунду.

Если не выбран игрок, выбираем.
Отнимаем ему хп.
Если добил, удаляем из живых.
Если прошла минута, пересчитываем nextMassAttackTime и добавляем inflictedDamage в BossMassAttack.

Если прошло две минуты, вытаскиваем из накопителя всех новых юзеров в топ и бъём сильнейших

Если реализовать это, можно встретить проблемы, которые не были видны на бумаге.

Задача 2

В игре есть MySQL-таблица, в которой хранится каждое действие игрока + какие-то данные о действии. В сутки в таблице собирается 10 миллионов записей. Требуется сделать просмотр этой статистики по этой таблице в виде отчета. Минимальный интервал отчета – сутки.

Вопросы

1. Как организовать просмотр статистики по суткам, неделям и месяцам, чтобы отчет строился в течении нескольких секунд?
2. Как обеспечить, чтобы действиям игроков не мешал просмотр статистики?

Реляционная база данных избыточная по функциональности для этой задачи. Поэтому и медленее, чем более специализированные решения. Из бесплатных решений есть Yandex ClickHouse.
Данные надо агрегировать по суткам, чтобы отчеты строились быстро. При агрегации нужно использовать запросы с установленными лимитами, чтобы не подвешивать БД. Желательно данные хранить в отдельной БД.

Задача 3

Есть игра, в которой данные о платежах хранятся в MySQL-базе данных. Диск на котором располагалась база благополучно сгорел.

1. Опишите возможные решений для исключения такой проблемы;
2. Выберите оптимальное решение и обоснуйте выбор;
3. Назовите несколько способов восстановления потерянных данных в результате такой поломки.

Облака, RAID-массивы, бэкапы+логи, софтовые и аппаратные решения.
Оптимальным будут облака - легко расширяются и администрируются и стоят скорее всего дешевле, чем альтернативы.