Міністерство освіти і науки,
молоді та спорту України
Національний Технічний
Університет України
«Київський Політехнічний Інститут»
Навчально-науковий комплекс
«Інститут прикладного системного аналізу»
Кафедра системного проектування

Лабораторна робота №5

З курсу: «Проектування інформаційних систем»

На тему: «**Модульне тестування (Unit-тести) та рефакторинг**»

Виконав:

Студент 4
курсу

Групи
ДА-72

Кондратю

к Тарас

Мета роботи: оволодіти навичками створення програмного забезпечення за методологією TDD та ознайомитися з процедурами рефакторинга.

Завдання:

1. Використовувати методологію Test Driven Development для створення класів архітектурної програмної моделі.

2. Скласти тестові сценарії, які продемонструють функціювання всіх методів проектованої моделі.

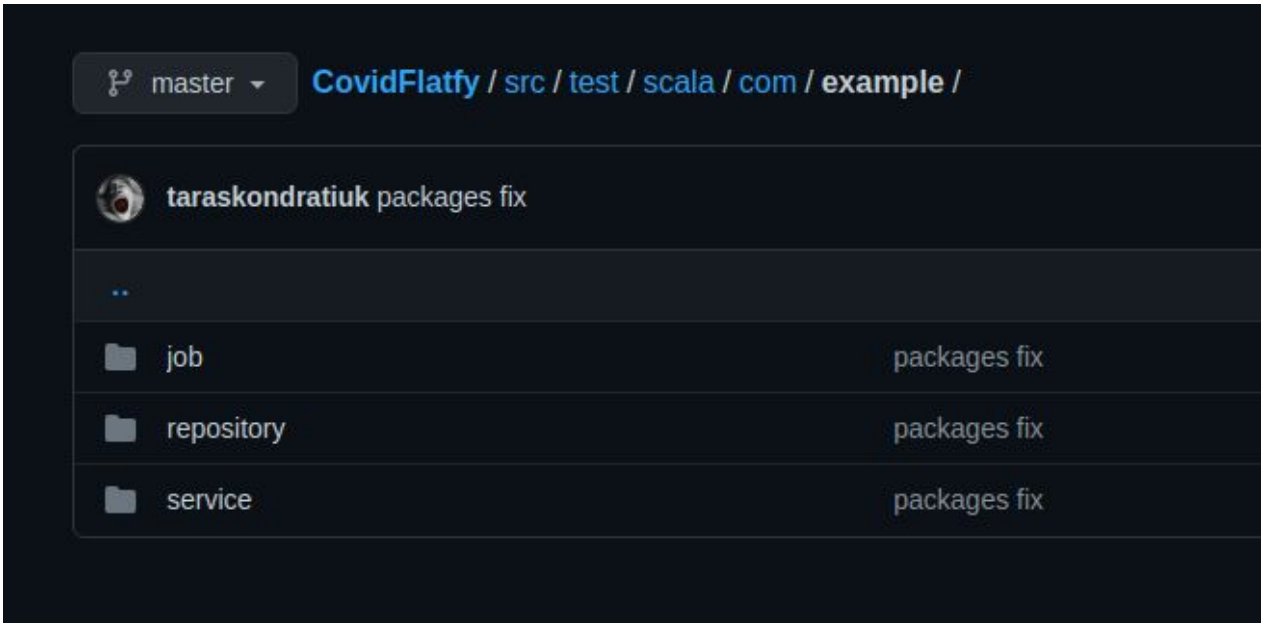
3. Виконати юніт-тестування складових частин (внутрішніх класів), що реалізують об'єкт моделювання.

4. Виконати "зовнішнє" юніт-тестування для API.

5. Провести рефакторинг коду програми, для поліпшення реалізації.

Хід виконання лабораторної роботи:

Розділив тести по директоріям з такими ж назвами, як шари системи.



Тести з директорії job перевіряють роботу cronjob-ів

```
test("job should download file") {
  val path = sys.env("PROJECT_PATH") + "\\src\\test\\resources\\jobtestdata.csv"

  new File(path).delete()
  val job = JobBuilder.newJob(classOf[DownloadAndRefreshCovidDataJob]).withIdentity("Job", "Group")
  val jobDataMap = new JobDataMap()
  jobDataMap.put("uri", sys.env("COVID_DATA_URI"))
  jobDataMap.put("path", path)
  job.setJobData(jobDataMap)

  val trigger: CronTrigger = TriggerBuilder.newTrigger()
    .withIdentity("Trigger", "Group")
    .withSchedule(CronScheduleBuilder.cronSchedule("0/10 * * * * ?"))
    .forJob("Job", "Group")
    .build

  val scheduler = new StdSchedulerFactory().getScheduler

  val covidCasesService = CovidCasesService()
  scheduler.start()
  scheduler.getContext.put("covidCasesService", covidCasesService)
  scheduler.scheduleJob(job.build(), trigger)
  Thread.sleep(15000)
  assert(new File(path).exists())
  assert(covidCasesService.kyivCovidCasesMap.toSet.nonEmpty)

}
```

завантажується файл і через 15 секунд викликається перевірка на його

існування

У директорії repository лежать тести що перевіряють рівень взаємодії з бд.

```
test("getLastListing should work properly") {
  val listing = Seq(RealEstateWithCovidCases("iii", "", "\"iii\"", "sdf", 3, Set("3", "волад")))

  val dateFormat = new SimpleDateFormat("yyyy:MM:dd:hh mm")
  val date = new Date()

  val lastKey = repo.r.get("lastKey")
  repo.saveListing(listing)
  val newListing = repo.getLastListing()
  assert(listing == newListing._2)
  repo.r.del(dateFormat.format(date))
  repo.r.set("lastKey", lastKey)
}
```

до репозиторію завантажуються дані, потім вони ж дістаються і порівнюються з завантаженими

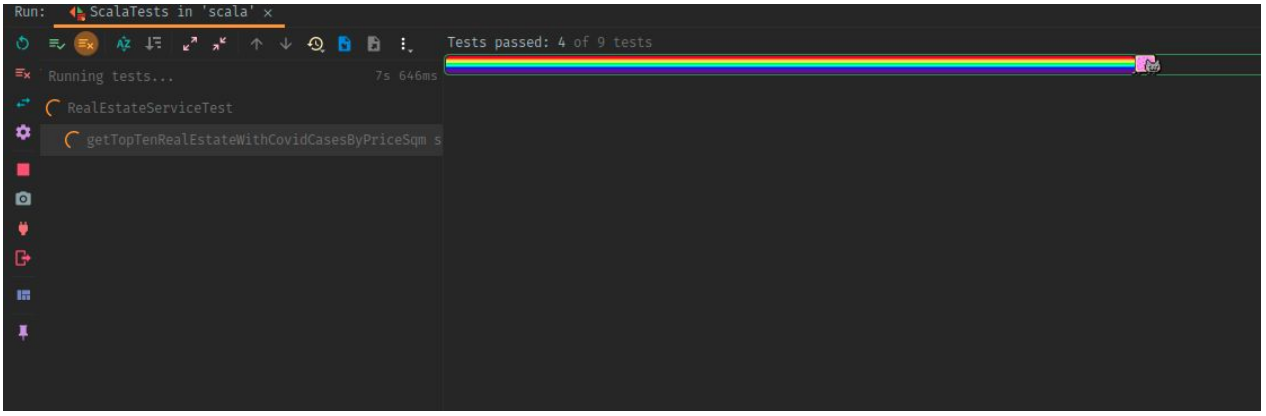
У директорії service знаходяться тести сервісів

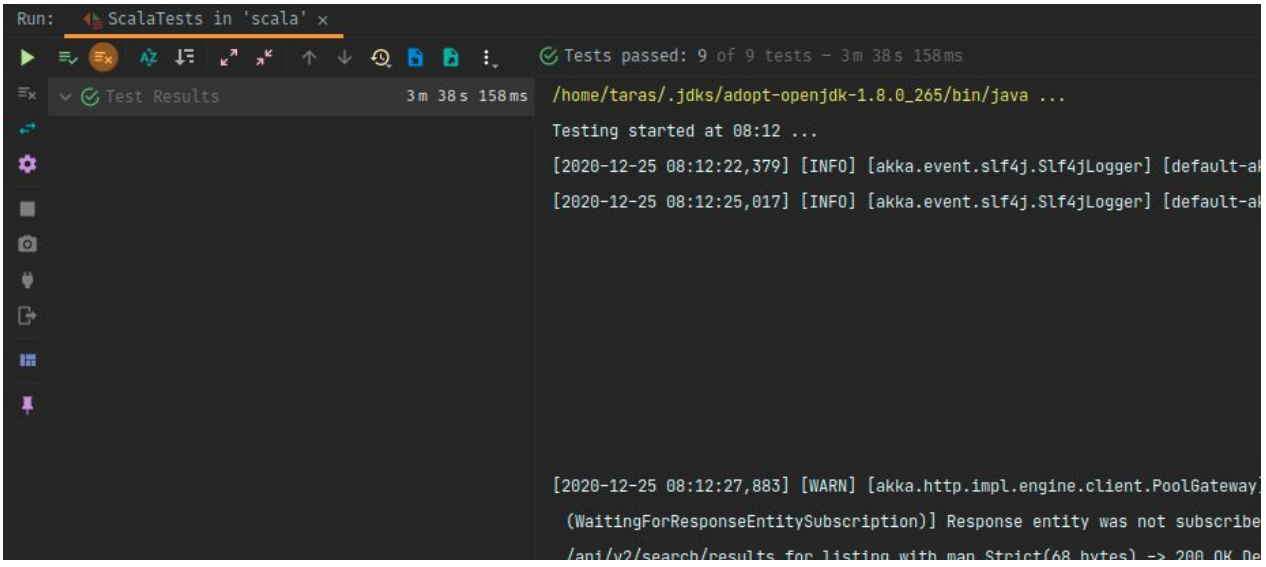
```
test("FileDownloader should download file") {
  val path = sys.env("PROJECT_PATH") + "\\data\\data.csv"
  new File(path).delete()

  downloadFile(sys.env("COVID_DATA_URI"), path)
  assert(new File(path).exists())
}
```

тут наприклад тестується метод downloadFile

Результат роботи:





Приклад TDD:

Спочатку був розроблен тест (перевірка умови, що api повертає 200):

```
test("api call should return status 200") {
  val future = service.getFlatfyResponseFuture(1)
  Await.result(future, Duration.Inf)
  assert(future.value.get.get.status.isSuccess())
}
```

Вже після нього код:

```
def getFlatfyResponseFuture(page: Int): Future[HttpResponse] = {
  Http().singleRequest(flatfyRequest(page))
}

private def flatfyRequest(page: Int) = HttpRequest(
  method = HttpMethods.POST,
  uri = sys.env("FLATFY_URI"),
  entity = HttpEntity(ContentTypes.`application/json`, flatfyRequestBody(page).compactPrint)
)

private def flatfyRequestBody(page: Int): JsObject = JsObject(
  "searchParams" -> JsObject(
    "city" -> JsNumber(1),
    "page" -> JsNumber(page),
    "realtyType" -> JsNumber(1),
    "contractType" -> JsNumber(1)
  )
)
```

Рефакторинг

Основним об'єктом для рефакторингу став RealEstateService, оскільки в ньому відбувається декілька перетворень над даними для отримання результатів. Зрештою постарався якомога сильніше розбити його на маленькі методи для спрощення читання та тестування.

Висновок

У ході виконання лабораторної роботи було розглянуто створення програмного забезпечення за методологією TDD та ознайомлено з процедурам рефакторинга.

Як мені здалося, TDD є досить ефективною методологією, що дозволяє в першу чергу зрозуміти завдання за допомогою власних тестів, а потім вже поисати код. Проте для майстерного освоєння цієї техніки необхідно багато часу, оскільки для роботи по тдд терба

досконало знати інструменти розробки й непогано розбиратися в області розробки. Також доволі складно перевчитися писати код як пару тестів -> один маленький метод і так по колу, це доволі незвично