Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Петрозаводский государственный университет»

Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

Федоренко Екатерина Алексеевна

Отчет по курсу «Верификация программного обеспечения»

Отчет о тестировании приложения “Valamis Learning Experience Platform”

Направление 09.04.02 — Информационные системы и технологии
Программа «Управление данными»

Преподаватель: к.ф.–м.н., доцент К.А. Кулаков

Петрозаводск, 2021

1. Объект тестирования

1.1. Описание программы

В настоящей работе рассматривается работа над платформой цифрового обучения **Valamis Learning Experience Platform** (LXP). Программный комплекс представляет собой веб-портал, который помогает инструкторам создавать учебные курсы, хранить весь контент в одном центральном месте в упорядоченном виде по категориям, делиться курсами со своей аудиторией и отслеживать их эффективность. Также портал включает в себя элементы социального обучения, такие как ленты активности, комментарии, отметки «Нравится» и публикации, которые помогают как преподавателям, так и учащимся совместно работать над курсом и делиться любыми ключевыми достижениями со всеми. Valamis LXP предоставляет пользователям доступ к углубленной аналитике в процессе обучения, поддерживая стандарт xAPI и встроенное хранилище учебных записей (LRS).

Valamis LXP — это платформа для обучения с открытым исходным кодом, которая помогает предприятиям создавать курсы и делиться ими со своей аудиторией в целях обучения и обучения.

рис. 1. Связь Valamis LXP с технологиями

1.2 Архитектура программы

Valamis LXP имеет открытый исходный код и построен на платформе портала Liferay. **Liferay Portal** — программный продукт, представляющий собой корпоративный портал, то есть решение, предназначенное для централизованного доступа к нескольким различным корпоративным приложениям в одном месте. Liferay иногда описывается как система управления содержимым (CMS) или платформу для веб-приложений.

Liferay Portal разработан на Java и работает на любой вычислительной платформе в среде Java Runtime Environment и сервере приложений. Liferay доступен в комплекте с сервером приложений Apache Tomcat.

Liferay Portal позволяет пользователям настроить общий доступ к разным приложениям через один-единственный сайт. Это реализуется с помощью функциональных модулей, называемых портлетами. Как и многие веб-приложения, портлеты обрабатывают запросы и генерируют ответы. В ответ портлет возвращает содержимое (например, HTML, XHTML) для отображения в браузерах.

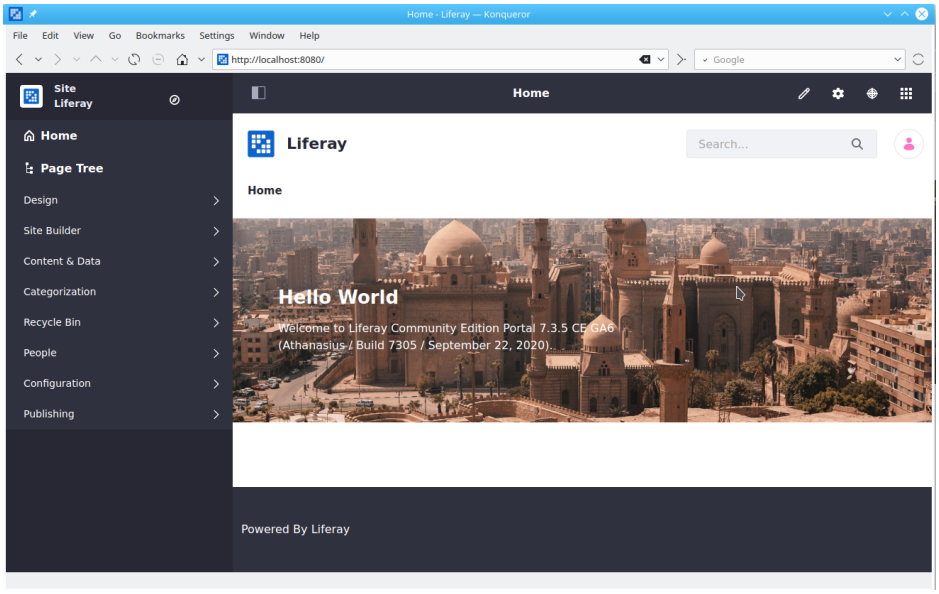
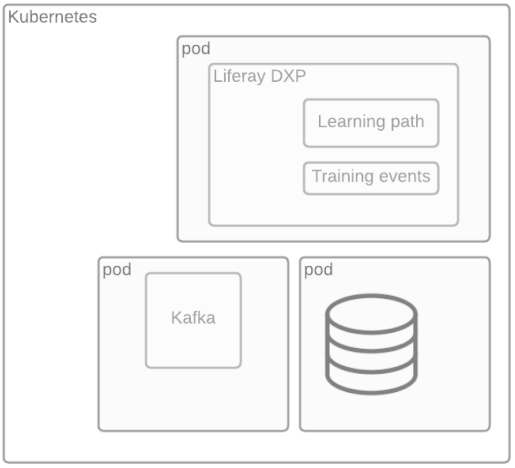


рис. 2. Страница портала Liferay 7.3.4

Работа программного продукта Valamis LXP состоит из связи огромного множества технологий, таких как Kubernetes, Kafka, LRS, Scala, Kotlin и многие

другие. В рамках учебного курса невозможно протестировать все из компонентов этого продукта, так как работой над ним занимаются различные команды. Малая часть архитектуры портала представлена на рисунке:



Пользователь портала может проходить множество “учебных путей” (Learning Path), участвовать в различных “тренировочных мероприятиях” (Training Events). Эти и многие другие сущности реализованы с помощью функционала портала **Liferay**. Непрерывную работу портала обеспечивает система Kubernetes, которая организует работу всех составных частей. Также связи между частями портала обеспечивает технология Kafka.
Работа учебных курсов основана на технологии LRS. Хранилище учебных записей (LRS) — это система хранения данных, которая служит репозиторием для учебных записей, собранных из подключенных систем, в которых проводятся учебные мероприятия.
Рассмотреть полноценное тестирование всех частей портала в рамках учебного курса невозможно, но мы можем выделить несколько “уровней”, в которых будем тестировать программу, и делать оценку ее работоспособности.

В настоящем отчете будет рассматриваться работа с системой Valamis LXP и его тестирование с различных сторон:

- внешний вид портала (портлет темы оформления)
- отправка данных в LRS,
- и работа с Kafka topic (аутентификация пользователей).

2. Стратегия тестирования

2.1. Структура объекта тестирования

Выбор объекта тестирования пал на несколько задач, которыми занимается команда интеграций, и те, которые влияют на непрерывную работу всего портала.

1) **Работа портлета ThemeSettings**, который отвечает за внешний вид портала и его части администрирования. Здесь мы будем имитировать работу браузера силами эмулятора Selenide.webdriver
2) **Отправление данных в Learning Record Store**. В этой части мы должны убедиться, что данные о курсах пользователей попадают в систему, а конкретно начало и конец завершения курса
3) **Модуль отслеживания пользовательской активности** (usertracking) - Работа с Kafka topics. Здесь мы будем проводить тестирование модуля, который отвечает за подключение пользователей к порталу.

2.2. Стратегия блочного тестирования

Блочные тесты проверяют работоспособность отдельных функций программы. Их поведение не зависит от результатов работы других тестируемых функций. Для их проверки в качестве аргументов при вызове функции передаются различные значения, после чего возвращаемый результат функции проверяется на наличие ошибок.

Блочное тестирование будет проводиться с помощью библиотеки Scalamock.

Liferay при авторизации юзера генерирует события, в частности onAfterUpdate, (событие после изменения данных пользователя) в данном случае меняется время последнего входа.
Код модуля должен следить за тем, чтобы после возникновения такого события информация о юзере попадала в Kafka и мы тестом это проверяем, что после

вызова события onAfterUpdate данные о пользователе попадают в определенные топики (в один в данном случае) и что эти данные такие, как ожидаются.

1) Создать мок сервер, это вынужденная мера, так как ServiceContextThreadLocal не существует в тестовом окружении, она есть только на реальных серверах
2) Сделать мок-сервер, который имитирует ее присутствие и если вызываются какие-то методы, он отвечает например так:

```
(request.getHeader _).expects("host").returns(realmId).anyNumberOfTimes()
(ctx.getRequest.getSession(_:
Boolean)).expects(true).returns(session).anyNumberOfTimes()
(session.getId _).expects().returns(testUserSessionId).anyNumberOfTimes()
```

3) Далее описывается, что если произойдет вызов метод getHeader, то ответить надо значением realmId

```
(request.getHeader _).expects("host").returns(realmId).anyNumberOfTimes()
```

4) Далее проверяется, что тестовый топик до записи в него пустой

```
userLoginTrackingProducer.history() shouldBe empty
```

5) Далее вызывается 2 события

```
userListener.onBeforeUpdate(testUser)
userListener.onAfterUpdate(testUser)
```

6) И проверяется, что данные в топике кафки теперь те, что отправлены

```
userLoginTrackingProducer.history() shouldBe Seq(
  KafkaProducerRecord(
    topic = "tracking.*******.liferay.user-login",
    key = UserIdWithCompanyId(testCompanyId, UserId(testUserId)),
    value = UserLogin(
      realmId = realmId,
      userId = Some(testUser.getUserUuid),
      sessionId = Some(testUserSessionId),
      ip = testUserIp,
      tpe = tpe
    )
  )
).asJava
```

2.3. Стратегия интеграционного тестирования

При интеграционном тестировании проверяется взаимодействие существующих модулей. Описанные выше, в блочном тестирование, функции, принимают ввод и отдают вывод другим. Таким образом проверяется отсутствие ошибок во взаимодействие их между друг другом.
Тестирование будет проводиться с помощью библиотеки Scalamock. Будет проверяться корректность отправки данных из программы (стейтментов) в систему LRS.

2.4. Стратегия аттестационного тестирования

Тестирование работы программы в целом. Производится путем запуска тестировщиком скомпилированной программы и созданием необходимых конфигурационных файлов. Таким образом проверяется работоспособность программы в виде приближенном к реальным условиям эксплуатации.

Тестирование будет проводиться методом автоматизированного тестирования с использованием фреймворка Selenide. Selenide - это "обертка" для Selenium Webdriver, ориентированная, как и Geb, на быструю и лаконичную автоматизацию тестирования, но написанная на Java. Преимущества этого инструмента:

- Удобный и лаконичный API
- Доступ к Webdriver
- Селекторы в стиле JQuery
- Работа с AJAX элементами
- Автостарт и уничтожение браузера

В этих тестах мы будем проверять итоговую работу пользователя с порталом, в частности будем имитировать некоторые его действия:

1) Открытие раздела “настройки пользователя”
2) Открытие раздела “закладки”
3) Открытие бокового меню портала

2.5. Критерии прохождения тестов

Тест считается пройденным если полученный и ожидаемый результат совпадают. Тестирование считается пройденным если во время его прохождения не выявлено критических ошибок и процент пройденных тестов не меньше 80%.

2.6. Критерии приостановки тестирования

Тестирование должно быть остановлено если количество непройденных тестов больше 30% от общего количества, а также при обнаружении критических ошибок сильно влияющих на функциональность приложения.

2.7. Критерии возобновления тестирования

Тестирование возобновляется при исправлении ошибок на предыдущем этапе тестирования.

3.1. Детальный план тестов

3.2. Блочное тестирование

Тест Б-1
Тип теста: общий
Описание: Тест проверяет, что при событии `onAfterRemoveAssociation` данные не падают в топики юзеров, но падают в `userAssociationTrackingProducer`
Входные данные: тестовый набор данных пользователей

Ожидаемый результат: метод вернет 1, если данные о пользователе попадают в определенные топики и что эти данные такие, как мы ожидаем

3.3. Интеграционное тестирование

Тест И-1 (взаимодействие LXP → LRS)
Тип теста: общий
Описание: тест проверяет, что отправленный в LRS statement имеет правильный формат согласно спецификации ISO 8601
Входные данные: тестовый набор данных с различными форматами даты стейтмента
Ожидаемый результат: метод вернет 1, если все введенные форматы соответствуют спецификации

3.4. Аттестационное тестирование

Тест А-1
Тип теста: общий
Описание: тест проверяет, что настройки пользователя открылись (проверка по заголовку, урл, характерным признакам страницы присущие только странице с настройками, что они отображаются)
Входные данные: открывается курс, клик по кнопке пользовательское меню, клик в открывшемся меню кнопку “настройки пользователя”
Ожидаемый результат: приложение предоставит пользователю экран “настройки пользователя”

Тест А-2
Тип теста: общий
Описание: проверяет, что страница с закладками открылась (проверка по заголовку, урл, характерным признакам страницы присущие только странице с букмарками, что они отображаются)
Входные данные: открывается курс, клик по кнопке “букмарки” (закладки)
Ожидаемый результат: приложение предоставит пользователю экран “закладки”

Тест А-3

Тип теста: общий

Описание: проверяет, что открылось продакт меню (проверка по появившимся вэбэлементам, списку курсов и тд)

Входные данные: клик по кнопке в боковом слайде “продакт меню”

Ожидаемый результат: приложение предоставит пользователю экран с открытым меню

4. Пример реализации теста

В качестве примера рассмотрим реализацию интеграционного теста И-1

```
import org.scalatest.prop.TableDrivenPropertyChecks
import org.scalatest.{ Matchers, PropSpec }

class DateTimeValidatorTest extends PropSpec with TableDrivenPropertyChecks with
Matchers {

 val dateTimes =
   Table(
     ("duration", "isRight"),
     ("2008-09-15T15:53:00.601", true),
     ("2011-07-15T00:00:00.000Z", true),
     ("2012-03-11T00:00:00+00:00", true),
     ("2012-03-11T00:00:00.301+07:00", true),
     ("2012-03-11T00:00:00.301-23:00", true),
     ("2012-03-11T00:00:00.301-07:00", true),
     ("2012-03-11T00:00:00.301-00:01", true),
     ("2012-03-11T00:00:00.301-00:50", true),
     ("1994-11-05T08:15:30-05:00", true),
     ("1994-11-05T08:15:30-15:00", true),
     ("2012-03-11T00:00:00+00:00", true),
     ("2008-09-15T15:53:00.601-01:00", true),
     ("2021-06-22T00:00+03:00", true),
     ("2011-07-15T00:00Z", true),
     ("2008-09-15T15:53:00.601-00", false),
     ("2008-09-15T15:53:00.601-0000", false),
     ("2008-09-15T15:53:00.601-00:00", false)
   )

 property("should validate date time") {
   forAll(dateTimes) { (data, isRight) =>
```

```
      dateTime(data).isRight shouldBe isRight
    }
  }
}
```

5. Журнал тестирования

Блочное тестирование

№	Дата	Тестировщик	Объект	Попытка	Результат
Б-1	10.12.2021	Федоренко Е.А.	`testUser`	1	Пройден

Интеграционное тестирование

№	Дата	Тестировщик	Объект	Попытка	Результат
И-1	10.12.2021	Федоренко Е.А.	DateTimeValidator	1	Ошибка (Отчет № 1)

Аттестационное тестирование

№	Дата	Тестировщик	Объект	Попытка	Результат
А-1	10.12.2021	Федоренко Е.А.	ThemeSettingsPortlet	1	Пройден
А-2	10.12.2021	Федоренко Е.А.	ThemeSettingsPortlet	1	Пройден
А-3	10.12.2021	Федоренко Е.А.	ThemeSettingsPortlet	1	Пройден

6. Журнал найденных ошибок

Отчет № 1

```
info] Tests: succeeded 64, failed 1, canceled 0, ignored 0, pending 0
[info] *** 1 TEST FAILED ***
[error] Failed tests:
[error]         com.******.******.web.servlet.admin.******
[warn] There may be incompatibilities among your library dependencies; run
'evicted' to see detailed eviction warnings.
[info] compiling 24 Scala sources to
/Users/efedorenko/*****/liferay71/target/scala-2.11/classes ...
[info] Run completed in 22 milliseconds.
[info] Total number of tests run: 0
[info] Suites: completed 0, aborted 0
[info] Tests: succeeded 0, failed 0, canceled 0, ignored 0, pending 0
[info] No tests were executed.
[error] (****** / Test / test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 212 s (03:32), completed Oct 18, 2021 3:46:10 PM
```

7. Результаты

В ходе тестирования была выявлена 1 ошибка. Часть из найденных ошибок существенно влияют на работу системы, но могут быть быстро исправлены без больших затрат. После исправления найденных ошибок программа должна стать работоспособной.