Разработка системы комплексной автоматизации деятельности выпускающей кафедры вуза на основе сервис-ориентированной архитектуры

Г.В. Праскура
Факультет информационных технологий
Брянский государственный технический университет
Брянск, Россия
e-mail: gregorypraskura@hotmail.com

Д.В. Титарев
Факультет информационных технологий
Брянский государственный технический университет
Брянск, Россия
e-mail: titaryovdv@mail.ru

Аннотация[1]

В настоящей статье описывается необходимость в наличии системы автоматизации, приводится общая архитектура разрабатываемой автоматизирующей информационной системы, подчеркиваются ее новизна и академическая польза в рамках образовательного учреждения. Объяснен выбор тех или иных архитектурных решений в сравнении с доступными альтернативами. Обозначены цели, стоящие перед выпускающей кафедрой. Описаны основные алгоритмы бизнес-логики, позволяющие достичь поставленных целей. Продемонстрированы практические результаты, уже полученные в ходе работы, а также те, что находятся на стадии разработки либо в проекте.

В 2015-м году на кафедре «Информатика и программное обеспечение» Брянского государственного технического университета [1] руководством были определены задачи по автоматизации деятельности кафедры. Необходимость автоматизации обусловлена недостаточной эффективностью исполнения повторяющихся ежедневных задач в рамках деятельности кафедры, которые выполнялись вручную. В частности, учет посещаемости и успеваемости студентов, учет учебных и научных достижений обучающихся.

Таким образом, было принято решение о создании единой информационной системы для комплексной автоматизации деятельности кафедры. В ходе реализации проекта по автоматизации деятельности кафедры были определены основные стейкхолдеры, приоритетные задачи, а также произведен выбор общей архитектуры системы.

Было принято решение о создании информационной системы на основе сервис-ориентированной архитектуры, а именно с использованием такого типа программного обеспечения как сервисная шина предприятия [2]. В качестве преимуществ данного подхода следует отметить следующее:

- Изучение новых технологий вне рамок курса обучения. В ходе проекта у студентов кафедры появилась возможность принять участие в его реализации и связать свои курсовые или выпускные квалификационные работы с созданием системы.
- Поддержка различных технологий. Промежуточное связующее программное обеспечение класса сервисных шин предприятия позволяет интегрировать сервисы, построенные на разных технологиях за счет унифицированных форматов обмена данными. Так как студенты и преподаватели могут иметь различные компетенции по технологиям, то и разработку вести они предпочтут с использованием разных технологий. Шина позволяет не фокусироваться на какой-то одной технологии, что положительно влияет на развитие компетенций обучающихся.
- Непосредственные преимущества самой сервисной шины предприятия как средства интеграции для автоматизации процессов на предприятии. Выпускающую кафедру вуза в данном случае можно рассматривать как предприятие – здесь ведется работа и с персоналом, и с документами, также присутствует большое количество учетных задач.

Общий вид архитектуры разработанной информационной системы приведен на рис. 1.

Труды Шестой всероссийской научной конференции "Информационные технологии интеллектуальной поддержки принятия решений", 28-31 мая, Уфа-Ставрополь, Россия, 2018

На представлении архитектуры информационной системы видно, что помимо Enterprise Service Bus (ESB-шина) также были развернуты: сервер репозиториев GitLab [3] и продукты компании Atlassian [4] – таск-трекер Jira [5] и Confluence [6].

Использование этих продуктов предусмотрено учебным планом кафедры, тем самым решая задачи проекта автоматизации, и, предоставляя незаменимый практический опыт студентам, задействованным в проекте. В данный момент времени эти сервисы имеют около ста активных пользователей.

Jira используется для моделирования командной работы над проектами в рамках курса «Введение в программную инженерию» у бакалавров первого курса специальности «Программная инженерия».

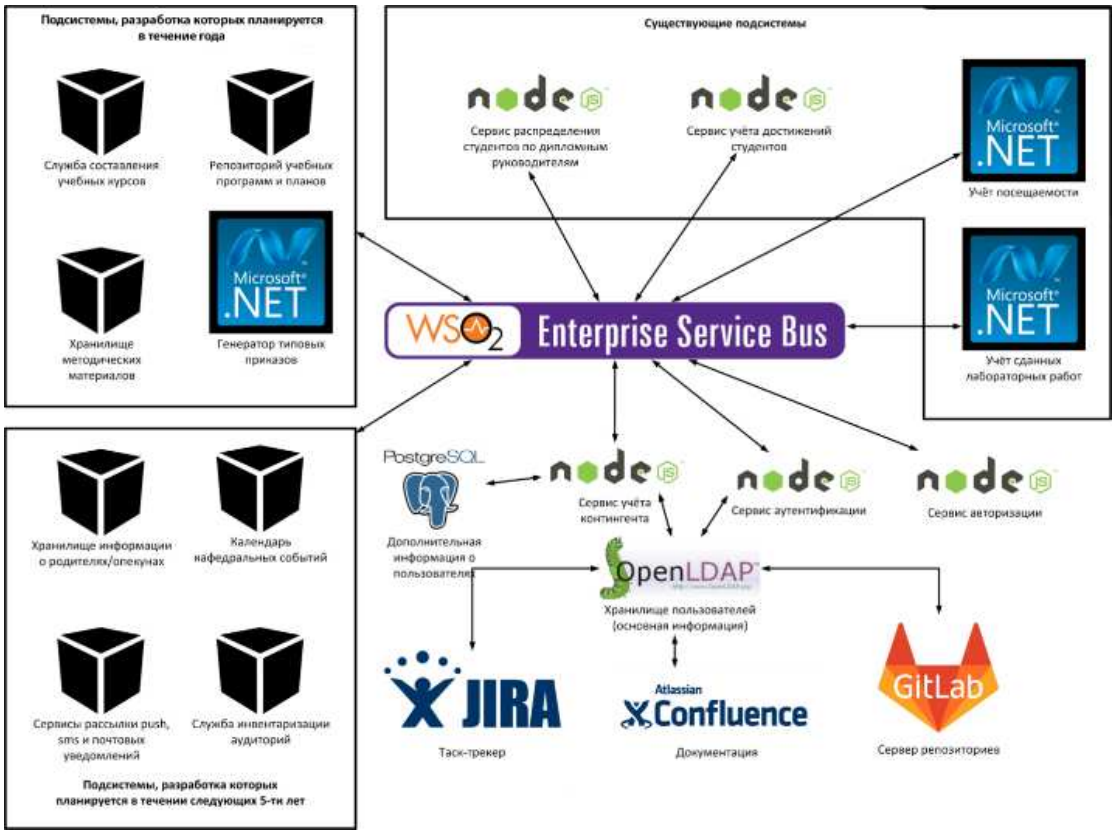

Рис. 1. Архитектура информационной системы

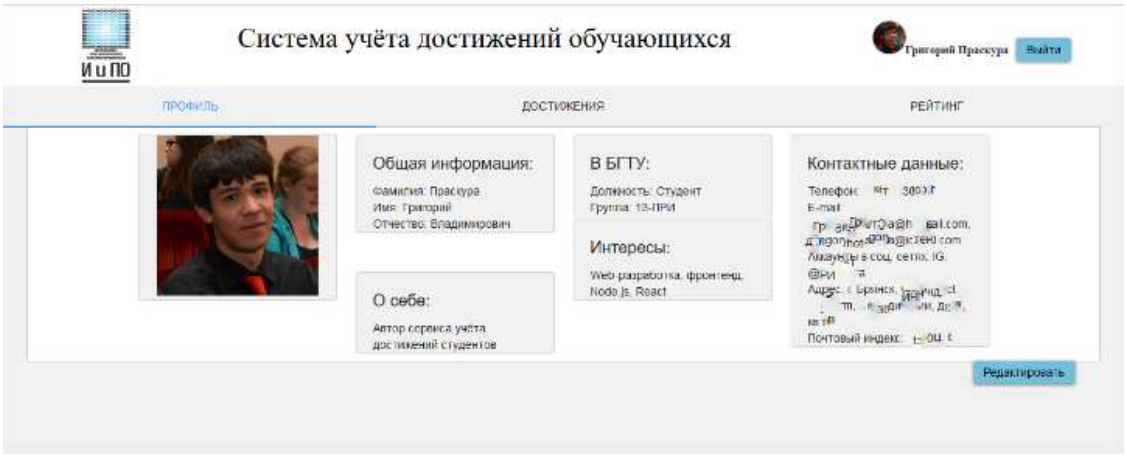

Рис. 2. Профиль сервиса учета достижений обучающихся

На первом этапе реализации проекта были созданы базовые сервисы:

- контингент,
- аутентификация,
- авторизация.

После создания базовых сервисов команда проекта приступила к разработке прикладных сервисов. Один из важных прикладных сервисов – это сервис учета достижений обучающихся.

Сервис учета достижений имеет связь со всеми базовыми сервисами и выполняет задачу по учету достижений обучающихся. Сервис размешен на собственном сервере кафедры [7]. Снимок экрана страницы профиля в сервисе учета достижений обучающихся представлен на рис. 2.

Веб-приложение реализовано с помощью библиотеки React.js [8], серверная часть написана на языке JavaScript [9] с использованием Node.js [10].

Одной из важнейших задач, стоящих перед выпускающими кафедрами, является распределение студентов по дипломным руководителям. При ее решении должны быть учтены научные интересы руководителей дипломного проектирования, квалификация студентов и пожелания к тематике дипломной работы.

В ходе выполнения магистерской диссертации одного из студентов кафедры было создано приложение для распределения студентов по дипломным руководителям.

Одной из приоритетных задач по развитию кафедральной информационной системы является модернизация сервиса контингента в части автоматической актуализации данных о студентах.

Для выполнения этой задачи был реализован следующий алгоритм. Кафедра получает от деканата файл с данными студентов (ФИО, электронная почта и группа). Процесс пересылки данных происходит с определенной периодичностью. Файл направляется на специальный почтовый ящик, который опрашивается сервисом синхронизации. По мере поступления в ящик новых файлов, сервис проводит процедуру синхронизации. А именно, вычисляется разница между предыдущим состоянием и новым, после чего эта разница применяется к текущему контингенту с сохранением истории.

Объекты истории содержат в себе информацию о перемещениях одного и того же обучающегося по разным группам, его отчислениях и восстановлениях, смене фамилии или других персональных данных, включая контактную информацию.

Ещё одной приоритетной задачей является автоматизация процесса назначения паролей пользователям информационной системы. Для аутентификации пользователей система использует протокол OpenLDAP [11].

Чтобы установить пользователю пароль, достаточно авторизоваться на LDAP-сервере с правами администратора и вручную указать пароль, зашифровав его. Вместо подобного неавтоматизированного подхода был предложен и реализован отдельный сервис, разбивающий зоны ответственности обучающихся.

Вначале администратор системы вводит временные пароли (генерируются автоматически) для старост групп обучающихся на кафедре. По инициативе старост происходит замена временного пароля постоянным, известным только владельцу. Далее старосты каждой из групп проводят аналогичные операции для остальных студентов их группы.

Таким образом, все обучающиеся получают возможность использования единой информационной системы. Так как в сервисе предусмотрено разделение прав по ролям, в представлении приложения доступны следующие возможности для каждой роли.

Администратор:

- Получение списка пользователей с указанием должностей (Студент, Староста, Преподаватель, Доцент, Заведующий кафедрой) и фильтрацией по ним.
- Для каждого пользователя возможность сгенерировать временный пароль. Если пользователь уже подтвердил его с помощью электронной почты или телефона, такая возможность отсутствует.
- Староста группы:
- Замена пароля (временный пароль имеет соответствующую метку) себе и подтвердить его по почте либо телефону.
- Получение списка пользователей, принадлежащих одной группе.
- Возможность назначить временный пароль студентам своей группы. При подтверждении пользователем староста больше такой возможности не имеет.

Студент:

- Замена пароля себе и подтверждение его по электронной почте или телефону.

В ходе проекта было реализовано веб-приложение для управления учетными записями, снимок экрана редактирования личных данных представлен на рис. 3.

Не менее важной задачей является создание тестовой среды для разработчиков. Эта задача решена с помощью технологии контейнеризации [12]: весь

сервер, на котором работают сервисная шина и базовые сервисы, можно упаковать в контейнер, создав docker-образ. Из этого образа возможно развертывание аналогичной среды по другому адресу.

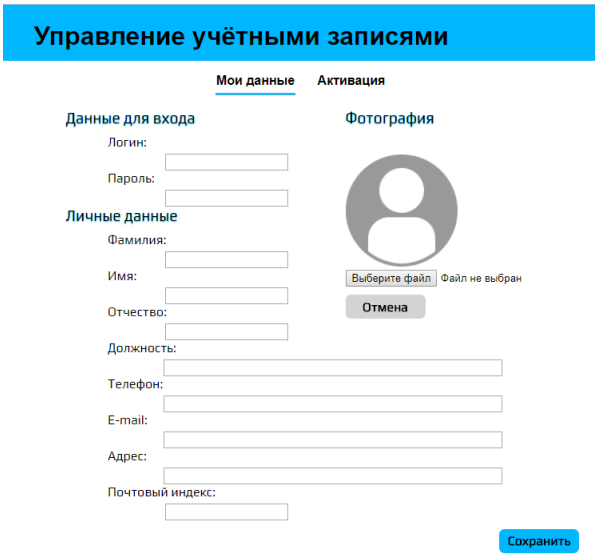

Рис. 3. Снимок экрана приложения управления учетными записями

С технологией контейнеризации также связана другая задача. Так как значительная часть выпускных и курсовых работ представляют собой веб-проекты, для их удобного развертывания необходимо создать отдельный сервис, который мог бы позволять обновлять, загружать и демонстрировать проекты.

В настоящее время наблюдается рост интереса к средствам виртуализации и облачных вычислений. Можно отметить спиралевидность истории: около полувека назад, когда централизованные вычисления были основной парадигмой компьютерных вычислений, IBM выпустила OS VM/370, позволявшую изолированно разделять мейнфреймы на программном и аппаратном уровнях для нескольких клиентов. Это положило начало активным исследованиям и разработкам в области виртуализации. И хотя контейнеризация ставит и выполняет отличные от виртуальных машин задачи, толчок ей дало развитие виртуализации.

Сейчас существует несколько продуктов, являющихся средствами контейнеризации: KVM, OpenVZ, LXC. Рассмотрим программные средства компании Docker по причине того, что именно они на текущий момент получают наиболее массовую поддержку среди разработчиков.

Docker – это компания, которая разрабатывает одноименную платформу для контейнеризации. Последнюю также называют «виртуализацией на уровне операционной системы». На рис. 4 показана разница между ней и классической виртуализацией.

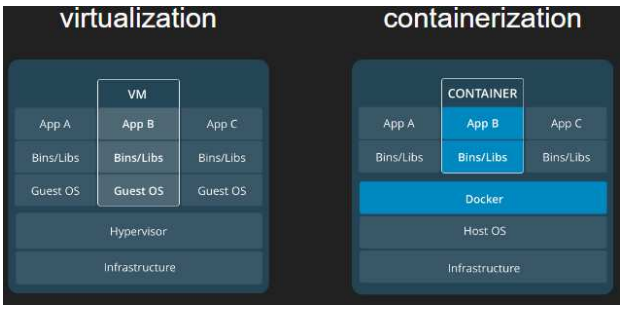

Рис. 4. Структурные отличия виртуализации и контейнеризации

Контейнеры схожи с виртуальными машинами тем, что они позволяют изолировать исполнение приложений в оперативной памяти, и приложения на диске выглядят как отдельная мини операционная система, поэтому каждый контейнер может обслуживаться собственными администраторами и иметь совершенно различных пользователей. Однако, в отличие от виртуальных машин, контейнер не нуждается в полноценном экземпляре операционной системы с ядрами, драйверами устройств и прочими зависимостями. Вместо этого используя единственный экземпляр операционной системы (хост), может исполняться целое множество контейнеров, использующее лишь долю ресурсов, которые потребовались бы полноценной виртуальной машине, выполняющей те же задачи.

Контейнеры наследуют зависимости от хоста контейнеризации. Это дает выгоды в плане управления системами, так как если на хост-системе контейнеризации были установлены исправления, все они будут унаследованы контейнерами.

Если для запуска полноценной виртуальной машины нужен гипервизор, то для работы контейнеров требуется хостовая операционная система или платформа контейнеризации, например, LXC с Docker. Именно по этой причине контейнеризацию также называют виртуализацией на уровне операционной системы.

Docker использует LXC – Linux Containers, расширяя его. Основными элементами для работы являются docker File, docker Image (образ), docker Volume (том). Dockerfile – это своеобразный скрипт, который используется для построения образа, который затем можно запускать в нескольких экземплярах.

Docker отличается тем, что предоставляет возможность сборки комплексных приложений и их загрузки в открытые хранилища – хабы, откуда их затем можно загружать в публичные или частные облака примерно так же, как вы загружаете приложения из App Store или другого публичного репозитория.

В рамках рассматриваемой идеи – создания сервиса для показа проектов важно отметить наличие и использование при работе с Docker интерфейса

командной строки – Docker command line. Именно команды Docker CLI будут передаваться из приложения сервиса на хостовую ОС контейнеров. Предполагается использование Node.js [10] и пакета node-ssh [14].

Можно выделить следующие преимущества описанного подхода:

- Скорость. Все работы по обслуживанию проектов за счет производительности Docker (написана на Go) происходят быстро.
- Безопасность. Изолированность проектов позволяет не заботиться об общих зависимостях.
- Низкий порог вхождения пользователей. Предполагается, что для пользователей (авторов проектов) процесс создания образов и контейнеров будет происходить прозрачно.

Заключение

Основные результаты, полученные в ходе работы над проектом, изложены ниже.

- Участниками проекта был получен опыт нестандартной для учебных программ разработки.
- Кафедра получила и активно использует в учебном процессе сервисы управления задачами и проектами Jira, ресурс для совместной работы Confluence, а также хранилище репозиториев GitLab.
- Создана основа для создания студентами продуктов для своих выпускных работ.
- Автоматизированы некоторые рутинные задачи, ранее производившиеся вручную.
- Развернута внутренняя тестовая среда, в которой студенты могут проводить эксперименты с участием своих проектов.

Список используемых источников

Список используемых источников оформляется согласно ГОСТ Р 7.0.5 2008 «Библиографическая ссылка»

1. Официальный сайт кафедры «Информатика и программное обеспечение» БГТУ — 2018. URL: http://iipo.tu-bryansk.ru (дата обращения: 11.03.2018).
2. Сервисная шина предприятия / Д.А. Шаппелл. — СПб.: БХВ-Петербург, 2008. — 370с.
3. Официальный сайт продукта GitLab. – 2018. URL: http://about.gitlab.com (дата обращения: 11.03.2018).
4. Официальная страница компании Atlassian – 2018. URL: https://ru.atlassian.com (дата обращения: 12.03.2018).
5. Официальная страница Atlassian Jira – 2018. URL: https://ru.atlassian.com/software/jira (дата обращения: 11.03.2018).
6. Официальная страница Atlassian Confluence – 2018. URL: https://ru.atlassian.com/software/confluence (дата обращения: 12.03.2018).
7. Система учета достижений студентов кафедры «ИиПО» БГТУ – 2017. URL: http://esb.iipo.tu-bryansk.ru/achievements (дата обращения: 11.03.2018).
8. Официальный сайт библиотеки React.js – 2018. URL: http://reactjs.org (дата обращения: 11.03.2018).
9. Документация по JavaScript – 2018. URL: http://developer.mozilla.org/ru/docs/Web/JavaScript (дата обращения: 12.03.2018).
10. Официальная документация Node.js – 2018. URL: http://nodejs.org/en/docs (дата обращения: 12.03.2018)
11. Понятия и обзор LDAP. — 2017. URL: https://pro-ldap.ru/tr/zytrax/ch2/ (дата обращения: 12.03.2018).
12. Использование Docker / Моуэт Э. — O'reilly Media, 2017. – 354с.
13. Документация пакета node-ssh. — 2018. URL: https://www.npmjs.com/package/node-ssh/ (дата обращения: 12.03.2018).