федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.04.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему Разработка облачной виртуальной лаборатории
для реализации математических моделей физических процессов

Студент Ю. В. Сизова

Научный руководитель Н. И. Лиманова

Руководитель магистерской программы доктор ф.-м.н., доцент, С.В. Талалов _____

«_____»_____20_____г.

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2017

Содержание

ВВЕДЕНИЕ

Математическое моделирование физических процессов – один из способов обучения, помогающий глубже разобраться в сущности различных явлений. Далеко не все учебные эксперименты можно или нужно проводить в реальных условиях. Технологии компьютерного моделирования достаточно быстро пришли в область естествознания. Работа не с самим явлением, а с его моделью дает возможность безболезненно, относительно быстро и без существенных затрат исследовать его свойства и поведение в любых ситуациях [5].

На сегодняшний день не так много русскоязычных информационных ресурсов, которые позволяют качественно проводить различные формы учебных занятий по физике в университете. В аудиториях не хватает лабораторного оборудования и с каждым днем растет потребность предоставлять дистанционное обучение [8], поэтому разработка виртуальной физической лаборатории с удаленным доступом является актуальной. Реализация виртуальной лаборатории с помощью облачных вычислений – технологии распределённой обработки данных, в которой компьютерные ресурсы предоставляются пользователю как интернет-сервис, поднимает вопрос о том, как применить данные технологии, чтобы качественно и быстро предоставить рабочую площадку каждому пользователю.

Объектом исследования магистерской диссертации является изучение подхода к разработке виртуальной физической лаборатории на основе облачных технологий.

Предметом исследования является процесс построения виртуальной лаборатории для моделирования физических процессов в виртуальной среде на основе облачных технологий.

Научная гипотеза заключается в том, что подход к разработке, основанный на облачных технологиях, напрямую влияет на скорость разработки и удобство использования виртуальной лаборатории, а также сокращает временные и материальные средства на разработку.

Новизна работы заключается в построении нового подхода для реализации инфраструктуры виртуальных лабораторий и расширении сферы применения виртуальной лаборатории благодаря разработке альтернативного способа загрузки виртуальных моделей.

Практическая значимость диссертации заключается в том, что технология виртуальной лаборатории, позволяющая загружать, использовать и редактировать виртуальные модели, дорогостоящая и мало распространена, тогда как разработанная виртуальная лаборатория останется бесплатной и может применяться не только для демонстрации, но и для создания визуального редактора моделей в обучающих целях.

На защиту выносятся следующие положения:

- подход к разработке облачной виртуальной физической лаборатории, который следует набору практик разработки программного обеспечения DevOps с использованием прикладного программного обеспечения для создания облачных технологий;
- расширение применения функциональных возможностей виртуальной лаборатории и обеспечение их доступности пользователям благодаря технологии загрузки виртуальных математических моделей в обобщенном виде.

Цель работы – создание подхода к разработке виртуальной физической лаборатории на основе облачных технологий и применение его для реализации виртуальной лаборатории с примерами математических моделей физических процессов. Для достижения цели были поставлены следующие задачи:

- выбрать физические процессы для моделирования;
- разработать математические модели физических процессов;
- спроектировать подход к разработке виртуальной лаборатории;
- выбрать средства разработки и обосновать выбор;
- реализовать загрузку виртуальных моделей;
- описать примеры математических моделей для инфраструктуры виртуальной среды;

- оптимизировать инфраструктуру виртуальной лаборатории;
- реализовать виртуальную лабораторию, используя выбранные средства на основе спроектированного подхода и требований к данной реализации;
- протестировать готовую реализацию и дать оценку результатам.

Таким образом, были приведены основные положения магистерской диссертации и необходимо раскрыть их в следующих главах.

ГЛАВА 1 АНАЛИЗ ИЗВЕСТНЫХ ВИРТУАЛЬНЫХ ФИЗИЧЕСКИХ ЛАБОРАТОРИЙ И МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ ФИЗИЧЕСКИХ ПРОЦЕССОВ

1.1 Виртуальная лаборатория и её место в образовательном процессе

Лаборатория – это объект, обеспечивающий контролируемые условия, в которых могут быть выполнены научные или технологические исследования, эксперименты и измерения [42].

Виртуальная лаборатория является интерактивным программным обеспечением на основе среды для проведения моделирования экспериментов. Лаборатория, главным образом, фокусируется на экспериментах, чтобы продемонстрировать теоретические концепции. Среда моделирования разрабатывается, чтобы передать чувство погружения, как будто студенты выполняют эксперимент в реальном мире. Эксперимент может быть реализован на основе прикладной программы автономного доступа или на основе веб-сервера и веб-браузера [10].

Основной целью разработки концепции виртуальной лаборатории дополнение реальной физической лаборатории в обучении. Основными задачами виртуальных лабораторий являются [40]:

- сократить расходы на обслуживание.
- дистанционный доступ к различным виртуальным лабораториям.
- мотивировать студентов к проведению экспериментов в собственных интересах.

Основная цель виртуальной лаборатории заключается в том, что пользователь может легко наращивать свои знания и совершенствовать применения фундаментальных концепций к практической работе.

В дополнение к удаленному доступу виртуальные лабораторные можно также дистанционно настраивать, что позволяет каждому отдельному студенту предоставить необходимые задания и виртуальные модели, с их настройкой по мере необходимости для конкретных практических занятий.

Виртуальная лаборатория может быть спроектирована различными способами в зависимости от программного курса, для которого она предназначена. Согласно исследованию [42], в случае умелого проектирования лаборатория должна обладать такими характеристиками, как реконфигурируемость, масштабируемость, рентабельность, надежность, поддерживаемость, реалистичность.

Лаборатория должна быть очень гибкой и повторно конфигурируемой. Разные темы и задания требуют различные модели. Необходимо разрабатывать и добавлять новые.

Лаборатория должна быть масштабируемой и должна быть в состоянии поддерживать многих студентов. Студенческие группы не должны получаться большими из-за отсутствия ресурсов.

Стоимость установки и обслуживания лаборатории должна быть гораздо меньше, чем то, что моделируется в лаборатории. Например, физические процессы, которые требуют большое количество аппаратуры.

Лаборатория должна быть в состоянии выдержать и обрабатывать непреднамеренные ошибки студентов в действиях во время неправильного проведения лабораторной работы.

Лаборатория должна быть простой в обслуживании. Обычные задачи, такие как резервное копирование и применения обновлений должны быть легко выполнимыми и автоматизированными до такой степени, до какой возможно.

Лаборатория должна иметь практическую значимость и основываться на законах из предметной области.

В научной статье [14] автор предлагает разделять ВУЛ в методологическом плане на три типа: процедурная, декларативная и гибридная (процедурно-декларативная). ВУЛ процедурного типа основана на учебных пакетах прикладных программ с интерфейсом, помогающим обучению. Студент должен знать математический аппарат при использовании данной лаборатории. Декларативная ВУЛ использует виртуальные кабинеты с подробным описанием технических объектов, подкрепленным картинками,

схемами аудио и видео материалом. Гибридный способ создания ВУЛ обычно используют при построении виртуальных приборов. При этом виртуальные объекты имеют такой же интерфейс, как и реальные приборы, а различные режимы работы исследуются с помощью математических или имитационных моделей [14].

Автор другой статьи [15] для всех виртуальных лабораторий выделяет общую схему их построений. Присутствие некоторых компонентов, по словам автора, не обязательно. Блок-схема структуры виртуальной лаборатории представлена на рис. 1.1.

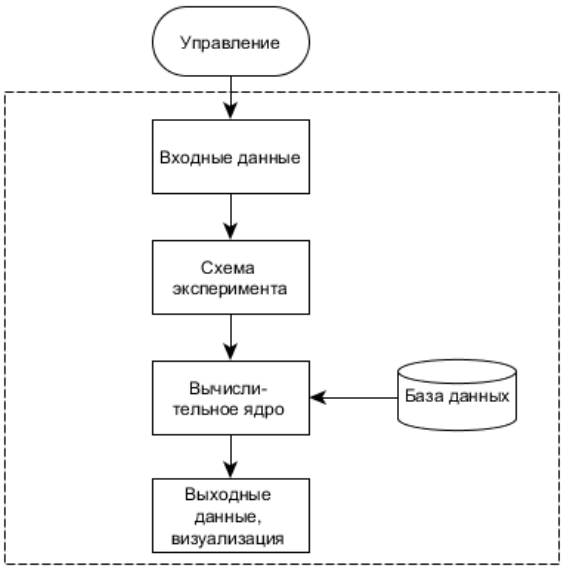

Рисунок 1.1 – Структура виртуальной лаборатории

Автор статьи [16] приводит подробный состав ВУЛ, обеспечивающий обучение студентов и контроль их знаний. Согласно методологическим исследованиям в виртуальную лабораторию должны входить:

- модуль имитации лабораторного эксперимента;
- модуль управления;
- модуль телекоммуникаций;
- модуль идентификации;
- обучающий модуль;
- модуль тестирования;

– модуль учёта транзакций;

– модуль визуализации;

– модуль ведения электронных образовательных ресурсов;

– модуль мониторинга.

Математическая модель такой ВУЛ будет выглядеть следующим образом: S – множество модулей, ответственных за имитацию лабораторных экспериментов, Z – множество иных модулей, R – множество связей между модулями, F – множество отношений базы данных (информационного обеспечения практикума). Работу в виртуальной лаборатории можно записать в виде кортежа $V = S\ Z\ R\ F$, $S = s \cup c$, где s – множество гиперссылок на виртуальные лаборатории, c – множество авторских лабораторий на сервере, $\cup$ – объединение множеств. Множество квантов теоретических знаний, требуемых для выполнения всех виртуальных лабораторных работ, $\kappa = \bigcup_{i=1}^{P(S)} K_i$, где P – мощность множества, K_i – множество порций знаний, требуемых для выполнения i-й работы. Виртуальную лабораторию представляют в виде $VL = \{BI\ SD\ SMM\ TS\ SCE\ OB\ SCVL\ LN\}$. Где BI – базовая информация о лабораторной работе, SD – множество условий, настроек и параметров проведения виртуального эксперимента и связанных с ними данных о результатах реальных экспериментов, SMM – множество мультимедийных материалов, соответствующих лабораторной работе, TS – множество дидактических задач по соответствующей теме, SCE – множество сценариев работы опыта, OB – множество виртуального лабораторных приборов и их описаний, $SCVL$ – множество сценариев работы виртуальной лаборатории, LN – множество связей лабораторной работы с элементами теоретического курса и соответствующими им составляющими электронных образовательных ресурсов.

В доказательство приведем исследование [44] о полезности виртуальной химической лаборатории. Работа показала, что виртуальные лаборатории, по крайней мере, столь же эффективны, как реальные лаборатории в плане

ознакомления студентов с процессом эксперимента, предоставляя студентам в безопасной экспериментальной среде индивидуально проводить эксперименты. Пользователям имеют больше возможностей в короткие сроки взаимодействовать одновременно с микро, макро и символическими уровнями представления [44]. Педагогический опыт показывает, что внедрение в образовательный процесс цифровых электронных технологий является сложным синергетическим явлением организации новой социокультурной учебной среды, где преподаватели и студенты готовы к творческой деятельности в ориентированных на профессию электронных цифровых средах [11].

1.2 Облачные технологии и их применение

Удаленные доступ к виртуальной лаборатории требует предоставлять и распределять сервисы виртуальной лаборатории между пользователями и администраторами.

Облачные вычисления – это один из видов интернет-технологий, который предоставляет общие компьютерные ресурсы обработки и данные для компьютеров и других устройств по запросу [12]. Данная модель обеспечивает повсеместный доступ по требованию к общему пулу конфигурируемых вычислительных ресурсов (например, компьютерных сетей, серверов, хранилищ, приложений и сервисов), которые могут быть быстро подготовлены и выпущены с минимальными затратами на управление [47]. Облачные вычисления и решения для хранения данных предоставляют пользователям и предприятиям различные возможности для хранения и обработки своих данных либо в частных, либо в сторонних центрах данных, которые могут быть расположены далеко от пользователей, как в пределах одного города, так и по всему миру. Облачные вычисления основаны на совместном использовании ресурсов для достижения согласованности и эффекта масштаба, подобно полезности сети электроснабжения [46].

Сторонники данной технологии утверждают, что облачные вычисления позволяют компаниям избежать предварительных расходов на инфраструктуру (например, приобретение серверов). Кроме того, такие вычисления позволяют организациям сосредоточиться на своем основном бизнесе, а не тратить временные и денежные ресурсы на компьютерную инфраструктуру [46]. Сторонники также утверждают, что облачные вычисления позволяют предприятиям быстрее запускать свои приложения с улучшенным менеджментом и меньшим обслуживанием. Также позволяют командам информационных технологий быстрее адаптировать ресурсы для удовлетворения неустойчивого и непредсказуемого бизнес-спроса [46].

В 2009 году наличие высокопроизводительных сетей, недорогих компьютеров и устройств хранения данных, а также широкое внедрение аппаратной виртуализации, сервис-ориентированной архитектуры и автономных вычислительных процессов привели к росту облачных вычислений [46]. Компании могут масштабироваться по мере роста потребностей в вычислительной технике, а затем снова уменьшаться по мере снижения требований. В 2013 году сообщалось, что облачные вычисления стали востребованной услугой, благодаря преимуществам высокой вычислительной мощности, дешевизне услуг, высокой производительности, масштабируемости, доступности [47]. Некоторые поставщики облачных решений увеличили темпы развития до 50% в год, но, находясь в начальной стадии развития, стакиваются с препятствиями, которые необходимо решить, чтобы сделать облачные вычислительные службы более надежными и удобными для пользователей [47].

Хотя сервис-ориентированная архитектура предоставляет «все как услугу» (с акронимами EaaS или XaaS или просто aas), поставщики облачных вычислений предлагают свои «услуги» в соответствии с различными моделями, из которых три стандартные модели для Национального института стандартов и технологий: инфраструктура как услуга (IaaS), платформа как услуга (PaaS) и программное обеспечение как услуга (SaaS) [27] . Эти модели предлагают возрастающую абстракцию. Поэтому они часто изображаются как слои в стеке

(см. рисунок 1.2): инфраструктура, платформа и программное обеспечение как услуга, но не обязательно должно быть так устроено.

Например, можно обеспечить SaaS, реализованную на физических машинах (bare metal), без использования базовых уровней PaaS или IaaS, и наоборот, можно запустить программу на IaaS и получить к ней доступ напрямую, не оборачивая ее как SaaS.

Определение Национального института стандартов и технологий облачных вычислений определяет модели обслуживания следующим образом [27]:

– программное обеспечение как услуга (SaaS). Способность, предоставляемая потребителю, заключается в использовании приложений поставщика, работающих в облачной инфраструктуре. Приложения доступны из различных клиентских устройств либо через тонкий клиентский интерфейс, такой как веб-браузер (например, веб-почта), либо через интерфейс программы. Потребитель не управляет или не контролирует лежащую в основе облачную инфраструктуру, включая сеть, серверы, операционные системы, хранилище или даже отдельные возможности приложений, за исключением ограниченных пользовательских настроек конфигурации приложения.

– платформа как услуга (PaaS). Способность, предоставляемая потребителю, заключается в развертывании в облачной инфраструктуре приобретенных или созданных пользователем приложений, разработанных с использованием языков программирования, библиотек, служб и инструментов, поддерживаемых поставщиком. Потребитель не управляет и не контролирует лежащую в основе облачную инфраструктуру, включая сеть, серверы, операционные системы или хранилище, но имеет контроль над развернутыми приложениями и, возможно, настройками конфигурации для среды размещения приложений.

– инфраструктура как услуга (IaaS). Способность, предоставляемая потребителю, заключается в обеспечении обработкой, хранении, сетями и другими фундаментальными вычислительными ресурсами, когда потребитель

может развертывать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы и приложения.

Рисунок 1.2 – Структура облачных технологий

Потребитель не управляет лежащей в основе облачной инфраструктурой, но имеет контроль над операционными системами, хранилищами и развернутыми приложениями. Управление выбором сетевых компонентов (например, межсетевых экранов) ограничено.

Основные характеристики облачной инфраструктуры представляют собой набор следующих характеристик.

Самообслуживание по требованию. Потребитель облачной службы может в одностороннем порядке и автоматически предоставлять вычислительные возможности, такие как серверное время и сетевое хранилище, по мере необходимости, не требуя человеческого взаимодействия с каждым поставщиком услуг.

Широкий доступ к сети. Способность потребителя облачной службы получать доступ к облачным ресурсам из любого места, используя широкий спектр устройств.

Объединение ресурсов. Вычислительные ресурсы поставщика объединяются, чтобы обслуживать нескольких потребителей с использованием модели с несколькими арендаторами, при этом динамические назначения и переназначение физических и виртуальных ресурсов динамически зависят от потребительского спроса.

Быстрая «эластичность». Возможности могут быть гибко обеспечены и выпущены, в некоторых случаях автоматически, чтобы быстро масштабироваться вовне и внутрь, соразмерно спросу.

Измеренный сервис. Облачные системы автоматически контролируют и оптимизируют использование ресурсов, используя возможности измерения на некотором уровне абстракции, соответствующей типу службы (например, хранение, обработка, пропускная способность и активные учетные записи пользователей).

Следует отметить, что в случае общедоступных облаков решающее значение имеет широкий сетевой доступ к облаку. Широкий доступ к сети позволяет облаку получать доступ через стандартные механизмы, которые способствуют использованию гетерогенных тонких или толстых клиентских платформ (например, мобильных телефонов, планшетов, ноутбуков и рабочих станций). Частные облака могут нуждаться или не требовать поддержки широкого сетевого доступа.

В статьях [6, 39, 9, 21] авторами рассматривается создание виртуальных лабораторий. Независимо от специфики лаборатории для предоставления пользователю вычислений используется несколько сервисов, связанных между собой по интерфейсу приложения. В статьях [6,35,36] авторы указывают, что пользователи вынуждены самостоятельно настраивать все необходимые им утилиты и рабочие инструменты для каждой задачи, причем большая часть этих инструментов работает только в операционной системе Linux. Поэтому

описывается расширяемая инфраструктура облачных технологий для предоставления публичного API и веб-сервиса для исследователей, и в рамках этой инфраструктуры предоставляется доступ к сервису.

1.3 Сравнение и анализ существующих виртуальных лабораторий

Интерактивные вычислительные образовательные ресурсы обычно разбросаны по сети. Более того, для их создания требуются специальные знания в области программирования, что делает их сложными и дорогостоящими. В результате их масштаб и охват ограничены. Перед разработкой необходимо исследовать особенности существующих виртуальных физических лабораторий.

Русскоязычный сервис Virtulab представляет собой набор образовательных интерактивных работ, позволяющих учащимся проводить виртуальные эксперименты по физике, химии, биологии, экологии и другим предметам, как в трехмерном пространстве, так и в двухмерном. Разработчики представляют свое программное обеспечение, как эффективное применение интерактивных тестов и уроков в образовательном процессе способствует не только повышению качества школьного образования, но и экономии финансовых ресурсов, создают безопасную, экологически чистую среду [19].

В свою очередь, основанный в 2002 году лауреатом Нобелевской премии Карлом Виманом, проект интерактивного моделирования PhET в Университете Колорадо создает бесплатные интерактивные математические и научные модели. Модели PhET основаны на обширных исследованиях в области образования и привлекают студентов через интуитивно понятную, игровую среду, где учащиеся учатся через исследования и открытия. PhET обеспечивает интересное, бесплатное, интерактивное, основанное на исследованиях научное и математическое моделирование. Каждая модель тщательно тестируется и оценивается, чтобы обеспечить эффективность обучения. Эти тесты включают интервью со студентами и наблюдение за использованием моделей в классах. Моделирование написано на Java, Flash или HTML5, и может быть запущено

онлайн или загружено на компьютер. Все симуляции с открытым исходным кодом. Несколько спонсоров поддерживают проект PhET, позволяя этим ресурсам быть свободными для всех студентов и преподавателей [49].

Wolfram demonstrations — это задуманный проект создателем Mathematica и ученым Стивеном Вольфрамом как способ довести вычислительное исследование до максимально широкой аудитории, проект Wolfram demonstrations представляет собой открытый код, который использует динамические вычисления для освещения концепций в науке, технике, математике, искусстве, финансах и других областей [51].

Ежедневно растущая коллекция интерактивных иллюстраций создается пользователями Mathematica со всего мира, которые принимают участие в содействии инновационным демонстрациям.

От начального образования до передовых исследований темы охватывают постоянно растущее множество категорий. Некоторые демонстрации могут использоваться для оживления классной комнаты или визуализации сложных понятий, в то время как другие проливают новый свет на передовые идеи из академических и промышленных рабочих групп. Каждый из них проверяется и редактируется экспертами для получения ясности, наглядности, качества и надежности.

Все демонстрации свободно распространяются на любом стандартном компьютере с операционными системами Windows, Mac или Linux. На самом деле, даже не нужен проект Mathematica. Можно взаимодействовать с любой демонстрацией, используя бесплатный Wolfram CDF PlayerЕсли установлена Mathematica, можно поэкспериментировать и модифицировать код самостоятельно.

Демонстрации могут быть созданы с помощью нескольких коротких строк кода. Это открывает двери для исследователей, преподавателей, студентов и профессионалов любого уровня для создания своих собственных сложных мини-приложений, а затем публикует их и делится ими с использованием формата вычисляемого документа Wolfram's (CDF).

Проект Wolfram demonstrations является частью семейства бесплатных онлайн-сервисов Wolfram Research – это Wolfram Alpha, первый в мире вычислительный вычислительный движок; MathWorld, сайт математики, а также сайты Wolfram, WolframTones и другие.

Важной уникальной характеристикой другой виртуальной лаборатории Molecular Workbench является то, что она воспроизводит имитационное моделирование в контексте обучения. Он поддерживает рендеринг и разработку учебной деятельности, которые, как правило, являются последовательностями шагов обучения, которые используют математические модели для обучения. Учебная деятельность может быть столь же простой, как демонстрация, которая показывает единую концепцию, или как полноценный, как цифровой учебник, который обеспечивает весь обучающий материал для темы или чего-то промежуточного [45]. Авторская система Molecular Workbench может быть использована для создания всех этих различных уровней учебной деятельности, подготовленной в классе, с настраиваемыми пользовательскими интерфейсами для разных студентов. Эта авторская система была использована для производства многих существующих видов деятельности, доступных через систему доставки МВТ, подобную браузеру. Molecular Workbench имеет больше функций, предназначенных для поддержки обучения и обучения. Это позволяет учителям создавать новые виды деятельности и/или изменять существующие. Все эти функциональные возможности, а также возможности моделирования и создания легко интегрируются в единую систему с простыми, интуитивно понятными графическими пользовательскими интерфейсами. Все эти вещи собраны вместе, размер загружаемого программного обеспечения составляет всего 1.6 Мб.

Виртуальная физическая лаборатория может использоваться различными способами, чтобы оживить преподавание физики:

- в качестве анимированной доски;

– в качестве виртуального эксперимента для получения типичных измерений данных;

– в качестве вспомогательного учебного плана;

– в качестве инструмента качественного исследования;

– как демонстрация того, как проводить эксперимент, чтобы позволить студентам управлять реальным физическим оборудованием в лаборатории;

– как инструмент для расширения знаний о предметах и углубления понимания учеников, а также учителей.

– в качестве справочной энциклопедии интерактивных экспериментов;

– как недорогой пакет регистрации данных (с использованием микрофонного входа звуковой карты и/или мыши в качестве датчика положения / угла).

Ресурс интуитивно понятен в своей работе, и каждое моделирование включает в себя инструкции, а также краткое описание физической теории. После установки на школьном сервере класс студентов в компьютерном классе может работать одновременно на любом количестве виртуальных моделей. Некоторые из анимаций и виртуальных экспериментов выходят за рамки того, что требуется в школе, но включены для расширения возможностей учеников. Виртуальная физическая лаборатория успешно используется в широком спектре учебных программ в разных странах [45].

Среди доступных лабораторий выделим качества, которые определяют удобство и функциональность программного продукта. Результат представим в таблице 1.1.

Таблица 1.1 – Сравнение виртуальных лабораторий

	wolfram	phet	virtulab	Molecular Workbench	VPLab
Возможность загрузки моделей из файла	+	-	-	+	-

Доступ онлайн	+	+	+	+	-
Контроль результатов работы	-	-	-	-	+
Технология для создания модели	wolfram language	Flash, Java Applet, HTML5	Flash	Java Applet	Java Applet
Русский язык	-	+	+	-	-
Свободное программное обеспечение	+	+	+	+	-

В настоящее время особенностью реализаций и использования виртуальных лабораторий является то, что если бесплатные системы позволяют собирать собственные эксперименты, то либо поделены на отдельные не связанные модули, либо модели не возможно редактировать пользователю сервиса, либо нельзя получать статистику о проделанных лабораторных работах в системе лаборатории, например Virtulab, Wolfram.

Заметно, что сама инфраструктура виртуальных лабораторий редко освещается в публикациях и в основном рынок программных решений составляют монолитные системы, с встроенными виртуальными моделями, которые не поддаются редактированию и расширению, как в случае с Virtulab, VPLab, Molecular Workbench.

Бизнес-решение VPLab [48], которое предоставляет полный пакет обучения и контроля на основе виртуальной лаборатории, платное и англоязычное.

В Molecular Workbench виртуальные физические модели создавались на основе платформы Java, в частности использовалась технология Java-апплет [45]. Она позволяла использовать один и тот же язык как для написания веб-

сервиса, так и для виртуальных моделей. Апплеты содержат большой выбор средств визуализации, создания экранных форм и поддерживают взаимодействие с пользователем. Все же, применение Java-апплетов имеет свои недостатки [20], которые затрудняют использование виртуальных моделей:

- требует установки Java-плагина для браузера;
- апплет не может быть использован, пока не запустится виртуальная Java-машина, запуск которой требует времени;
- в виду небезопасности апплетов, пользователи по умолчанию не смогут их запустить;
- зачастую требуется использование определенной среды выполнения Java.

Поэтому следует остановиться на создании моделей с использованием Java Script, так как данный язык реализации не требует дополнительных действий, кроме как использование современного браузера.

Данная ситуация делает вопрос о разработке информационной инфраструктуры, позволяющей в обобщенном виде загружать виртуальные модели с расчетом на их редактирование и возможность расширять сервисы виртуальной лаборатории, актуальным.

Перейдем к описанию виртуальных моделей, подлежащих разработке, и требований к ним.

1.4 Анализ виртуальных физических моделей, подлежащих разработке

На кафедре ТГУ «Общая и теоретическая физика» в данный момент используется «Открытая физика 2.6, часть 2» [20] для проведения лабораторных работ и моделирования физических процессов. Перечислим некоторые темы лабораторных работ, которые выполняются в рамках курса физики в виртуальной физической лаборатории:

- исследование движения электрона в электрическом поле;
- исследование двойного маятника;

– исследование обратного маятника на подвижной платформе.

Текущее программное обеспечение содержит в себе простые формулы без возможности неоднозначного подхода для исследования моделируемой системы. Кроме того, путем тестирования было установлено, что существующие виртуальные модели имеют много неточностей в предоставляемых данных. Необходимо смоделировать процессы, учитывая, по возможности, различные свойства, присущие реальной системе, и описать в виртуальной среде полученные математические модели в качестве примера использования инфраструктуры виртуальной лаборатории.

Виртуальная модель – это модель физического объекта, которая является цифровым описанием объекта (сильно упрощенного) [42].

Для первой модели на вход должны поступать следующие параметры: напряженность электрического поля, начальная скорость по x. На выход поступают скорость по y, время, напряженность при вылете из обкладок конденсатора. Размер, заряд частицы, длина обкладок конденсатора фиксированы. Модель анимирует движение частицы согласно введенным параметрам.

Для второй модели нужно регулировать длину двух плечей маятника, как и вес каждого подвеса, а также начальный угол отклонения для каждого плеча.

Для третьей модели должны задаваться параметры усиления (пропорциональная, интегрирующая и дифференциальная составляющие ошибки угла), состояния (положение, скорость, угловая скорость и угол) и свойства (длина маятника, вес груза, коэффициент трения, внешняя сила и вес тележки) моделируемой системы обратного маятника на тележке. Также время воспроизведения движения задается вручную. В итоге можно наблюдать колебания маятника на тележке в рамках заданных условий. Графически результаты работы отображаются диаграммой, показывающей изменения угловой скорости, угла отклонения маятника, положения и силы.

Согласно исследованиям автора статьи [38], для построения хороших математических моделей необходимо осуществить следующие этапы:

– определение проблемы для вычислительной модели. Это утверждение должно обеспечивать описание цели создания модели, вопросы, на которые она поможет ответить, тип ожидаемых результатов, значимых для этих вопросов;

– определение спецификации модели, чтобы помочь определить концептуальную модель решаемой проблемы. Это описание того, что получится с разрабатываемой вычислительной моделью, так же принятие на себя ответственности (ограничений) и соблюдающаяся область законов. В идеале спецификация модели должна быть точна, завершена, лаконична и понятна. Это описание включает список значимых компонентов, взаимодействие между компонентами и динамику поведения модели;

– определение математической модели. Этот этап включает получение представления решения проблемы с использованием математических сущностей и выражений и деталей алгоритма для отношений и динамики поведения модели;

– реализация модели. Выбор программного обеспечения для симуляции – так тоже важное практическое решение. Главные задачи на этом этапе – это программирование, отладка и тестирование программной модели;

– проверка модели. С помощью разных запусков реализации модели (или программы модели) на этом этапе сравнивают выходные результаты с теми, что были бы получены правильной реализацией концептуальных и математических моделей. Этот этап сосредотачивает внимание на попытке документировать и доказать верность реализации модели;

– оценка модели. На этом этапе сравнивают выходные данные проверенной модели с данными реальной системы (или похожей уже разработанной модели). На этом этапе сравнивают свойства и данные модели с доступными знаниями и данными о реальной системе. Оценка модели пытается дать заключение размеру улучшений для понимания модели.

Для наглядности моделей используем эвристики, предложенные автором статьи [43]:

– использовать модель, чтобы согласовать множественные формы представления.

– использовать модель, чтобы содействовать дискуссии.

– создавать ситуации, напоминающие игру, и получите пользу от явных и неявных задач.

– сфокусироваться на показательных случаях. Ученые часто говорят об исследовании граничных случаев, когда изучают проблему.

– попросить студентов заново воспроизвести или представить параметры на симуляции.

– использовать методы «предсказывай, наблюдай, объясняй».

Таким образом, был проделан анализ научных работ, связанных с разработкой структуры виртуальных лабораторий и с учетом существующих наработок решено создать виртуальную учебную лабораторию декларативно-процедурного типа на основе облачных технологий.

Далее перейдем к описанию математических моделей, имитирующих физические явления, для виртуальной физической лаборатории.

ГЛАВА 2 МАТЕМАТИЧЕСКИЕ МОДЕЛИ И ИНФРАСТРУКТУРА ВИРТУАЛЬНОЙ ФИЗИЧЕСКОЙ ЛАБОРАТОРИИ

2.1 Математическая модель движения частицы в однородном поле конденсатора

Математическое моделирование - это представление закономерностей поведения объекта исследования в виде математических соотношений. Математической моделью называют систему уравнений и неравенств, которые включают наиболее значимые характеристики исследуемого объекта или процесса и упрощенно отражают взаимосвязи между ними [2]. Математическая модель позволяет определить значения выходных и внутренних характеристик объекта или процесса по известным значениям входных характеристик с учетом возмущающих и управляющих воздействий без экспериментальных исследований [1].

В учебных целях обычно берется следующее описание частицы в однородном поле конденсатора. Однородным называется поле, напряженность которого во всех точках одинакова как по величине, так и по направлению [4]. Например, поле заряженного конденсатора однородно.

Сила, действующая на заряженную частицу в однородном поле, везде одинакова, поэтому неизменным будет и ускорение частицы, определяемое вторым законом Ньютона [4]:

$$\vec{a} = \frac{\vec{F}_{ЭП}}{\mathrm{m}} = \frac{q}{m}\vec{E} = const, \tag{2.1}$$

где q – величина заряда;

m – масса частицы;

E – напряженность поля.

Тогда скорость равноускоренного движения электрона вдоль оси у найдем по формуле:

$$V_y = at_{ДВ} = \frac{q}{m} E \frac{L}{V_{ox}}, \tag{2.2}$$

где V_Y – вертикальная составляющая скорости электрона в момент вылета из конденсатора;

V_{ox} – горизонтальная составляющая начальной скорости движения электрона вдоль оси x;

L – длина пластины конденсатора.

Величину пройденного электроном пути вдоль оси y определим по формуле [4]:

$$y = \frac{at_{ДВ}^2}{2} = \frac{q}{2m} E \left(\frac{L}{V_{0x}} \right)^2 . \tag{2.3}$$

Величину пройденного электроном пути вдоль оси x определим по формуле:

$$x = V_{0x} t_{ДВ} . \tag{2.4}$$

Скорость вылета электрона из конденсатора будет рассчитываться по формуле:

$$V_{вып} = \sqrt{V_{0x}^2 + V_{yt}^2} . \tag{2.5}$$

Данные формулы просты и предполагают, что краевыми эффектами, т.е. уменьшением напряженности поля по мере удаления от обкладок, пренебрегают. Внести приближение к реальным свойствам доступными методами задача следующей части.

2.2 Математическое моделирование краевых эффектов плоского конденсатора

Определение напряженности электрического поля в любой точке поля заряженного конденсатора с учетом краевых эффектов потребовало бы решение системы уравнений Максвелла [3]. Этот способ требует хорошей теоретической подготовки, значительных временных затрат для реализации и нуждается в ресурсах для вычислений, что оправдано лишь при рассмотрении

данного вопроса со студентами физических специальностей вузов. Поэтому для расчета краевых эффектов электростатического поля плоского конденсатора для студентов инженерного профиля нашего вуза мы предлагаем рассмотреть имитационную модель, позволяющую в той или иной мере учесть краевые эффекты.

В первой модели поле конденсатора предлагается заменить полем диполя, расположенного так, как показано на рисунке 2.1 (точки A и A').

Ось x расположена симметрично обкладкам конденсатора на расстоянии *d/2* от каждой обкладки. Ось y перпендикулярна оси x и направлена вертикально вверх. Модуль заряда диполя равен модулю заряда одной из обкладок заряженного конденсатора. Ширину обкладок считаем настолько тонкой, что ею можно пренебречь, причем, чем ближе к обкладке конденсатора расположены заряды диполя, тем больше искажается поле внутри конденсатора.

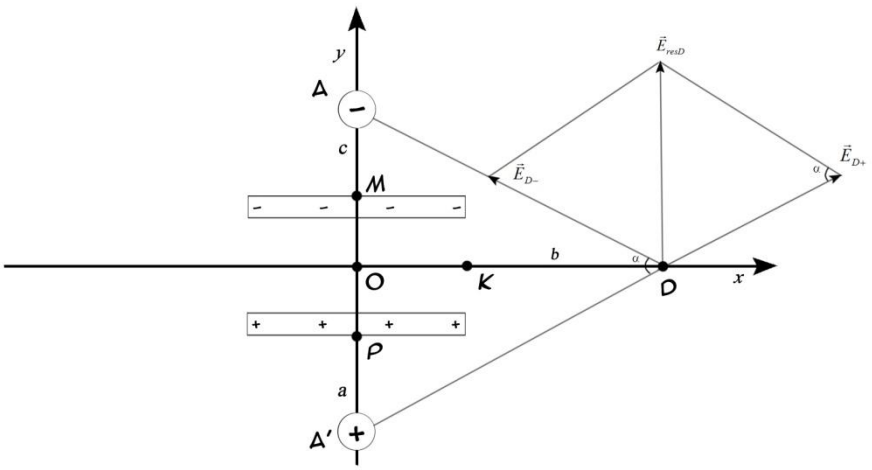

Рисунок 2.1 – Взаимное расположение обкладок конденсатора и диполя

Поэтому отрицательный заряд q_1 и положительный заряд q_2 располагаются на расстояниях a от обкладок конденсатора. Результирующая напряженность $\vec{E}_{resD}$ данной системы рассчитывается согласно принципу суперпозиции.

Из рисунка 2.1 видно, что $OA = \frac{d}{2} + a$, $OD = \frac{l}{2} + b$, $2OA = AA'$. Следовательно $A'D = \sqrt{OA^2 + OD^2}$.

Напряженность поля, созданного зарядом $q_1 d$ в точке D, равна:

$$\vec{E}_{resD} = \frac{kq_1}{AD^2} = \frac{kq_1}{OA^2 + OD^2}. \quad (2.6)$$

По теореме косинусов

$$E^2{}_{resD} = E_{D+}{}^2 + E_{D+}{}^2 - 2E_{D+}E_{D-}\cos\alpha = 2E_{D+}(1-\cos\alpha), \quad (2.7)$$

$$AA'^2 = 2A'D^2 - 2A'D^2\cos\alpha^2 = 2A'D^2 - 2A'D^2\cos\alpha, \quad (2.8)$$

$$\cos\alpha = \frac{(AA')^2 + 2(A'D)^2}{2(A'D)^2} = \frac{-\left[2(\frac{d}{2}+a)\right] + 2\left[(\frac{d}{2}+a)^2 + (\frac{l}{2}+b)^2\right]}{2(\frac{d}{2}+a)^2}. \quad (2.9)$$

Следовательно, окончательно получаем:

$$E_{resD} = \sqrt{2E_{D+}{}^2(1-\cos\alpha)}. \quad (2.10)$$

Следующая модель дает точное решение для расчета поля конденсатора, пластины которого бесконечны в одном направлении задачи. В основе метода расчета лежат два принципа: принцип возможности представления рассчитываемой величины в дифференциальной форме и принцип суперпозиции [3].

Каждую из обкладок заряженного конденсатора мы представляем в виде системы тонких полосочек, настолько тонких, что их можно представить как систему заряженных нитей конечной длины. Эти нити, несущие равнораспределенные по длине заряды разного знака, расположены симметрично оси x. Заряд одной нити можно рассчитать по формуле:

$$q_{нити} = \frac{q_{конденсатора}}{2N}, \quad (2.6)$$

где N – число нитей, на которые разделили одну пластину.

Величину линейной плотности заряда рассчитаем по формуле:

$$\lambda = \sigma \Delta x, \tag{2.7}$$

где σ – поверхностная плотность заряда.

Все нити, расположенные на нижней обкладке конденсатора равномерно заряжены с положительной линейной плотностью заряда, а нити верхней обкладки - с такой же по модулю, но противоположной по знаку линейной плотностью заряда [13]. Рассчитаем напряженность поля, созданного бесконечно длинной, равномерно заряженной нитью (рис.2.2)

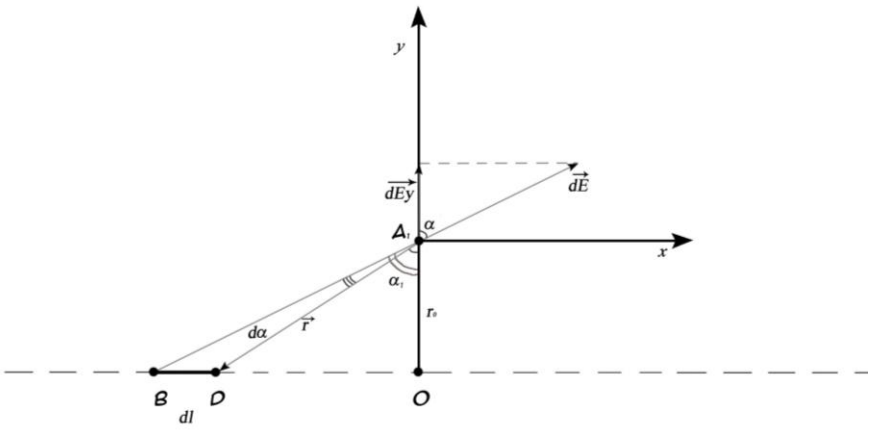

Рис. 2.2 – Расчет напряженности поля, созданного бесконечно длинной нитью, в точке A_1

Для бесконечно длинной нити, напряженность в точке A_1, равноудаленной от концов этой нити будет равна:

$$E_{y\,рез} = \frac{2\lambda}{4\pi\varepsilon_0 r_0}. \tag{2.8}$$

Эта значение совпадает с величиной напряженности поля плоского бесконечно длинного конденсатора, рассчитанного с помощью электростатической теоремы Гаусса.

Перейдем теперь к расчету напряженности поля, созданного нитью конечной длины. На рисунке 2.3 показано направление элементарного вектора напряженности поля в точке A_2, равноудаленной от концов нити конечной длины.

Ось x направим так, чтобы она делила наш заряженный конденсатор на две симметричные части и располагалась на расстоянии d/2 от каждой из обкладок конденсатора, а ось y направим перпендикулярно оси x вертикально

вверх. Точка A_2, для которой будем рассчитывать величину напряженности поля, может смещаться вдоль оси x.

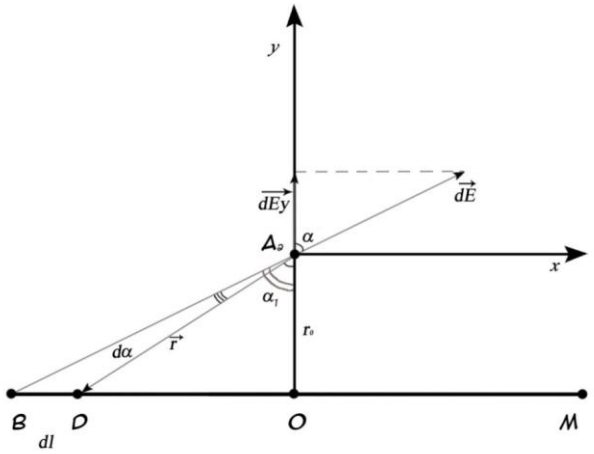

Рис. 2.3 – Расчет напряженности поля, созданного нитью конечной длины в точке A_2

Для расчета разделим нить на малые элементы dl, так чтобы $dl = \lambda dl$ и найдем величину напряженности поля, созданной одной равномерно заряженной ниточкой конечной длины.

$$dE = \frac{1}{4\pi\varepsilon_0}\frac{dq}{r^2} = \frac{\lambda dl}{4\pi\varepsilon_0 r^2}. \tag{2.9}$$

Из треугольника AOD следует, что $\frac{r}{r_0} = \cos\alpha$, соответственно, из треугольника DAB получается $dl = \frac{r_0 d\alpha}{\cos^2\alpha}$. Проекции $d\vec{E}$ на осях $0x$ и $0y$:

$$0x:\ dE_y = \frac{\lambda\cos\alpha d\alpha}{4\pi\varepsilon_0 r_0}, \tag{2.10}$$

$$0y:\ dE_x = \frac{\lambda\sin\alpha d\alpha}{4\pi\varepsilon_0 r_0}.$$

После интегрирования для общего случая получаем:

$$E_x = \frac{\lambda}{4\pi\varepsilon_0 r_0}(\cos\alpha_1 - \cos\alpha_2), \tag{2.11}$$

$$E_y = \frac{\lambda}{4\pi\varepsilon_0 r_0}\int_{\alpha_1}^{\alpha_2}\cos\alpha d\alpha = \frac{\lambda}{4\pi\varepsilon_0 r_0}(\sin\alpha_1 + \sin\alpha_2).$$

В силу симметрии расположения равномерно заряженных ниточек относительно оси Ox, суммарное значение $E_x=0$.

Остается рассчитать для точки A_2 только игрековую проекцию результирующей напряженности, которая будет в 2 раза больше $E_{y\ одной\ нити}$:

$$E_{y\ рез} = \frac{2\lambda}{4\pi\varepsilon_0 r_0}(\sin\alpha_1 + \sin\alpha_2)\,. \tag{2.12}$$

Синусы интересующих нас углов вычисляются по формулам:

$$\sin\alpha_1 = \frac{BO}{\sqrt{BO^2 + r_0^2}},\ \sin\alpha_2 = \frac{OM}{\sqrt{OM^2 + r_0^2}}\ . \tag{2.13}$$

При перемещении точки А вдоль оси х вправо или влево будет изменяться величина α_1 (см. рис. 2.3). При этом изменяются величины $\sin\alpha_1$ и $\sin\alpha_2$.

Следует учесть, что значение синуса α_2 в дальнейших расчетах следует брать со знаком минус.

Так в точке A_3, показанной на рис. 2.4, напряженность результирующего электрического поля будет равна:

$$E_{y\ рез} = \frac{2\lambda}{4\pi\varepsilon_0 r_0}(\sin\alpha_1) \tag{2.14}$$

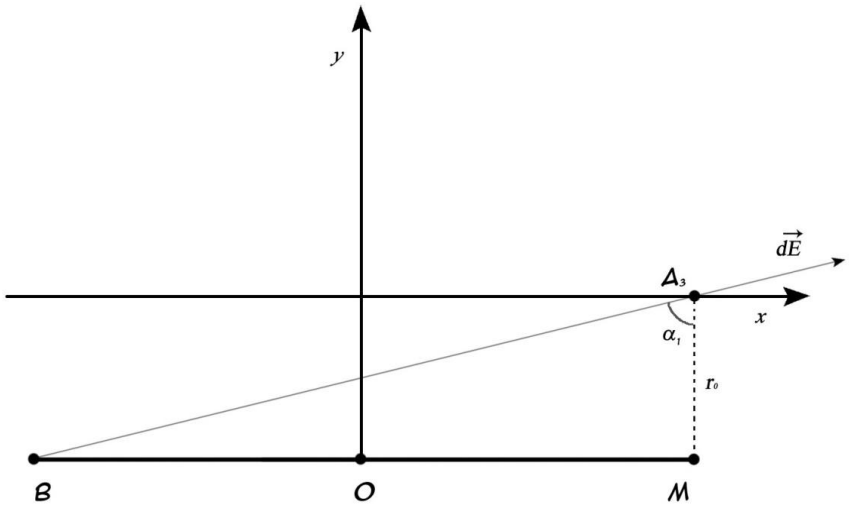

Рис. 2.4 – Расчет напряженности поля, созданного нитью конечной длины в точке A_3

В точке A_4 показанной на рис. 2.5, напряженность результирующего электрического поля будет равна:

$$E_{y\,рез} = \frac{2\lambda}{4\pi\varepsilon_0 r_0}(\sin\alpha_1 - \sin\alpha_2)\,. \qquad (2.15)$$

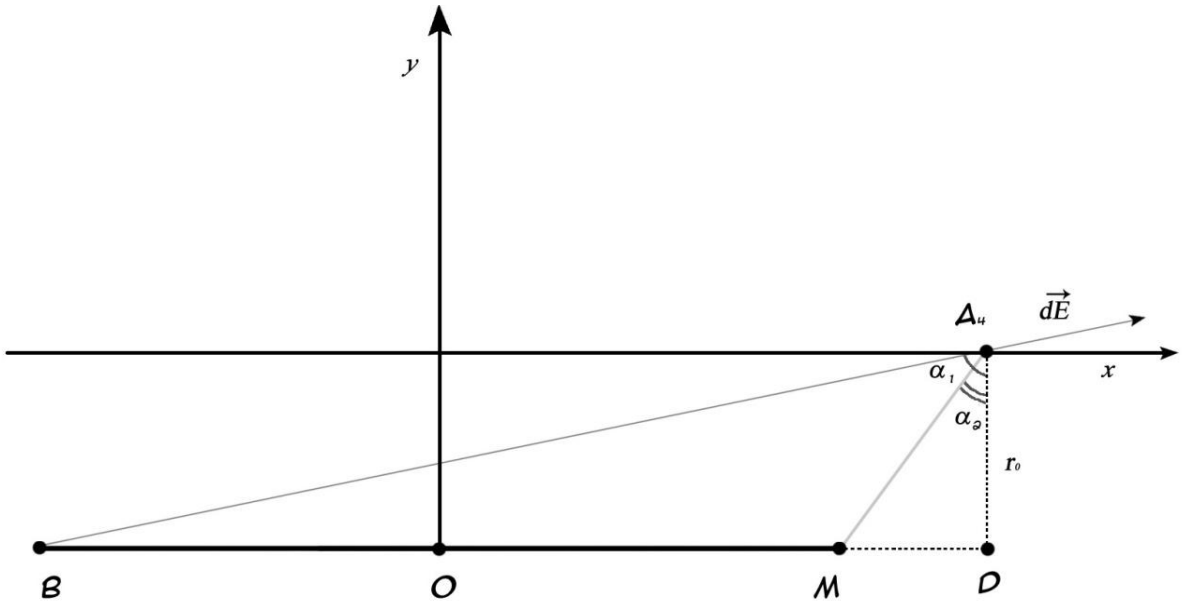

Рис. 2.5 – Расчет напряженности поля, созданного нитью конечной длины в точке A_4

Расчеты показывают, что величина напряженности поля в точке А при смещении ее от краев конденсатора вдоль оси х заметно уменьшается.

Таким образом предложенная модель позволяет рассчитать величину напряженности электростатического поля конденсатора и ее погрешность измерения при удалении точки А вдоль оси х от краев его обкладок, т.е. учесть краевые эффекты.

Проведем вычислений согласно модели краевых эффектов, основанной на представлении значения в виде дифференциальной формы.

Так как значение результирующей напряженности вне обкладок конденсатора в общем виде можно представить, как $Eres = \frac{2\lambda}{4\Pi\varepsilon_0} C$, тогда относительную погрешность метода рассчитаем в виде:

$$\Delta Eres = < \Delta Eres > \sqrt{\left(\frac{\Delta\lambda}{\lambda}\right)^2 + \left(\frac{\Delta r_0}{r_0}\right)^2 + \left(\frac{\Delta c}{c}\right)^2}, \quad (2.16)$$

где $< \Delta Eres >$ - значение напряженности на каждом шаге. Результат представим в таблице 2.1.

Таблица 2.1 – Относительная погрешность значений краевых эффектов

$< \Delta Eres >$	$\Delta Eres$
1390.868252	34.7717063
1389.619746	34.74049365
1387.957972	34.6989493
1385.885404	34.64713509
1383.405116	34.58512791
1380.520777	34.51301943
1377.236633	34.43091581
…	…
519.0127208109	12.97531802
511.2901682056	12.78225421
503.6805088800	12.59201272
496.1829140391	12.40457285

Слагаемые c и λ как разница синусов углов и линейная плотность малы и стремятся к нулю, поэтому ими можно пренебречь.

Остается рассчитать значение $\left(\frac{\Delta r_0}{r_0}\right)$ и умножить на результирующую напряженность через каждый шаг, равный $\Delta l = 0.0005$ м при $\Delta r_0 = 0.00025$ м, $r_0 = 0.01$ м.

Относительная погрешность уменьшается, как и уменьшается значение результирующей напряженности при удалении от края обкладок конденсатора.

Абсолютную погрешность представим как отношение промежуточного результата к относительной погрешности, умноженное на 100 процентов. Округленно для всех значений абсолютная погрешность составляет 2.5%.

2.3 Математическая модель двойного маятника

В физике и математике в области динамических систем двойной маятник представляет собой маятник с другим маятником, прикрепленным к его концу, и является простой физической системой, которая обладает богатым динамическим поведением с сильной чувствительностью к начальным условиям [3]. Движение двойного маятника определяется набором связанных обыкновенных дифференциальных уравнений и является хаотическим.

Чрезвычайно трудно предсказать, проявляя, казалось бы, случайное или хаотичное поведение. Движение двойного маятника можно смоделировать, используя систему обыкновенных дифференциальных уравнений. Однако, поскольку эти уравнения не имеют аналитического решения, они должны быть аппроксимированы численно. Это можно сделать с помощью компьютера с соответствующим образом написанной программой. Схематичное изображение двойного маятника представлено на рисунке 2.6.

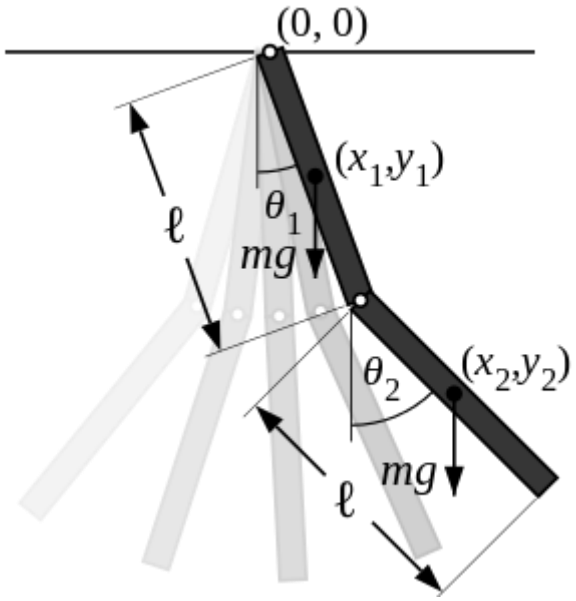

Рис. 2.6 – Схема двойного маятника

Можно рассмотреть несколько вариантов двойного маятника. Два плеча могут иметь равные или неодинаковые длины и массы, они могут быть простыми маятниками или составными маятниками (также называемыми

комплексными маятниками), и движение может быть в трех измерениях или ограничено вертикальной плоскостью. В следующем анализе плечи считаются одинаковыми составными маятниками длиной l и массой m , и движение ограничивается двумя измерениями.

В составном маятнике масса распределяется вдоль его длины. Если масса распределена равномерно, то центр масс каждого плеча находится в своей средней точке, а плечо имеет момент инерции $I = \frac{1}{12} ml^2$ относительно данной точки [4].

Удобно использовать углы между каждым плечом и вертикалью в качестве обобщенных координат, определяющих конфигурацию системы. Этими углами обозначены θ_1 и θ_2. Положение центра масс каждого стержня может быть записано в терминах этих двух координат. Если считать, что начало декартовой системы координат находится в точке подвеса первого маятника, то центр масс этого маятника находится в точке:

$$\begin{cases} x_1 = \frac{l}{2}\sin\theta_1 \\ y_1 = -\frac{l}{2}\cos\theta_1 \end{cases}. \tag{2.17}$$

И центр масс второго маятника находится в точке

$$\begin{cases} x_2 = -l(\sin\theta_1 + \frac{1}{2}\sin\theta_2) \\ y_2 = -l(\cos\theta_1 + \frac{1}{2}\cos\theta_2) \end{cases}. \tag{2.18}$$

Формул 2.17 и 2.18 достаточно, чтобы написать лагранжиан.

Лагранжиан равен разнице кинетической и потенциальной энергии.

$$\begin{aligned} L &= \frac{1}{2} m(v_1^2 + v_2^2) + \frac{1}{2} I(\dot{\theta}_1^2 + \dot{\theta}_2^2) - mg(y_1 + y_2) \\ &= \frac{1}{2} m(\dot{x}_1^2 + \dot{y}_1^2 + \dot{x}_2^2 + \dot{y}_2^2) + \frac{1}{2} I(\dot{\theta}_1^2 + \dot{\theta}_2^2) - mg(y_1 + y_2) \end{aligned} \tag{2.19}$$

.

В формуле 2.19 первый член - линейная кинетическая энергия центра масс тел, а второй член - вращательная кинетическая энергия вокруг центра масс каждого стержня. Последний член - потенциальная энергия тел в однородном гравитационном поле. Точечная запись показывает производную по времени от рассматриваемой переменной.

Подставляя координаты 2.8 и 2.9 и преобразовывая уравнение, получим:

$$L = \frac{1}{6} ml^2 [\dot{\theta}_2^2 + 4\dot{\theta}_1^2 + 3\dot{\theta}_1^2 \dot{\theta}_2^2 \cos(\theta_1 - \theta_2)] + \frac{1}{2} mgl(3\cos\theta_1 + \cos\theta_2). \quad (2.20)$$

Есть только одна сохраняемая величина (энергия) и не сохраняется обобщенный импульс. Два импульса могут быть записаны как

$$\begin{cases} p_{\theta_1} = \dfrac{\partial L}{\partial \dot{\theta}_1} = \dfrac{1}{6} ml^2 [8\dot{\theta}_1 + 3\dot{\theta}_2 \cos(\theta_1 - \theta_2)] \\ p_{\theta_2} = \dfrac{\partial L}{\partial \dot{\theta}_2} = \dfrac{1}{6} ml^2 [2\dot{\theta}_2 + 3\dot{\theta}_1 \cos(\theta_1 - \theta_2)] \end{cases}. \quad (2.21)$$

Эти выражения можно инвертировать, чтобы получить

$$\begin{cases} \dot{\theta}_1 = \dfrac{6}{ml^2} \dfrac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2) p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)} \\ \dot{\theta}_2 = \dfrac{6}{ml^2} \dfrac{8p_{\theta 2} - 3\cos(\theta_1 - \theta_2) p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)} \end{cases}. \quad (2.22)$$

Остальные уравнения движения записываются в виде

$$\begin{cases} \dot{p}_{\theta_1} = \dfrac{\partial L}{\partial \theta_1} = -\dfrac{1}{2} ml^2 [\dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) + 3\dfrac{g}{l} \sin\theta_1] \\ \dot{p}_{\theta_2} = \dfrac{\partial L}{\partial \theta_2} = -\dfrac{1}{2} ml^2 [-\dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) + \dfrac{g}{l} \sin\theta_2] \end{cases}. \quad (2.23)$$

Эти последние четыре уравнения являются явными формулами для временной эволюции системы с учетом ее текущего состояния. Невозможно пойти дальше и интегрировать эти уравнения аналитически, чтобы получить формулы для θ_1 и θ_2 как функции времени. Однако это число можно выполнить численно с использованием метода Рунге Кутта или аналогичных методов. Угловые ускорения для θ_1 и θ_2 описывают эволюцию движения маятника.

В уравнениях движения тригонометрические функции $\sin, \cos$ определяют состояние и далее берутся их производные. Это нелинейная система. В частном случае малых перемещений в качестве начальных условий уравнения движения можно положить $\theta_1(t=0)$, $\theta_2(t=0) \approx 0$, уравнение может быть упрощено с использованием приближения малых углов. Тогда, например, если взять другие частные случаи, такие как $m_1 \square m_2$ или $m_1 \square m_2$, то можно рассмотреть с аналитическими подходами, которые имеют приблизительное гармоническое решение; это также может быть определено аналитически.

Можно получить следующие уравнения движения

$$\begin{cases} \dot{\theta}_1 = \omega_1 \\ \dot{\omega}_1 = \dfrac{m_2 l_1 \omega_1^2 \sin\Delta\cos\Delta + m_2 g \sin\theta_2 \cos\Delta + m_2 l_2 \omega_2^2 \sin\Delta - Mg\sin\theta_1}{Ml_1 - m_2 l_1 \cos^2\Delta} \\ \dot{\theta}_2 = \omega_2 \\ \dot{\omega}_2 = \dfrac{-m_2 l_2 \omega_2^2 \sin\Delta\cos\Delta + M(g\sin\theta_1\cos\Delta - l_1\omega_1^2\sin\Delta - g\sin\theta_2)}{Ml_1 - m_2 l_1 \cos^2\Delta} \end{cases}, \quad (2.24)$$

где $\omega_{1,2}$ – угловой момент отклонения;

$\Delta = \theta_2 - \theta_1, M = m_1 + m_2$;

g – гравитационное ускорение.

Для моделирования двойного маятника необходимо решить уравнения 2.24 в терминах t. Поскольку не существует аналитического решения, оно должно быть выполнено численно, методов данных вычислений существует несколько. Наиболее простым из них является метод Эйлера, который утверждает, что при заданной функции $y(t)$, которая

$$y(t+\Delta t) = y(t) + \dot{y}(t)\Delta t, \quad (2.25)$$

где Δt – инкрементируемое время;

$\dot{y}(t)$ – производная $y(t)$ по времени t.

Кроме того, даны начальные значения $\theta_{1,2}(t)$ и $\omega_{1,2}(t)$, что делает возможным рассчитать модель двойного маятника численно [3].

2.4 Математическое моделирование обратного мятника на подвижной платформе

Перевёрнутый или обратный маятник представляет собой маятник, который имеет центр масс выше своей точки опоры, закреплённый на конце жёсткого стержня. Часто точка опоры закрепляется на подвижной платформе (тележке), которая может перемещаться по горизонтали, схематично данная система изображена на рисунке 2.7. В то время как нормальный маятник устойчиво висит вниз, обратный маятник по своей природе неустойчивый и должен постоянно балансироваться, чтобы оставаться в вертикальном положении с помощью применения крутящего момента к опорной точке или при перемещении точки опоры по горизонтали, как части обратной связи системы.

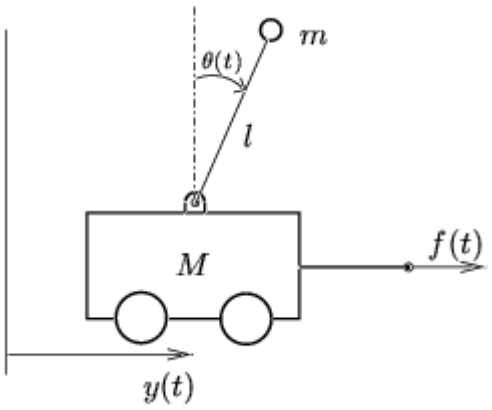

Рис. 2.7 – Схематическое изображение обратного маятника на тележке

Простейшим демонстрационным примером может являться балансировка карандаша на конце пальца.

Уравнения движения могут быть получены с использованием уравнений Лагранжа. Речь идёт рис. 2.7, где $\theta(t)$ угол маятника длиной l по отношению к вертикали и действующей силе гравитации и внешних сил F в направлении x. Определим $x(t)$ положение тележки. Лагранжиан $L = T - V$ системы:

$$L = \frac{1}{2}Mv_1^2 + \frac{1}{2}mv_2^2 - mgl\cos\theta, \tag{2.26}$$

где v_1 является скоростью тележки;

v_2 – скорость материальной точки m.

v_1 и v_2 могут быть выражены через x и θ путём записи скорости как первой производной положения.

$$v_1^2 = \dot{x} \tag{2.27}$$
$$v_2^2 = \left(\frac{d}{dt}(x - l\sin\theta)\right)^2 + \left(\frac{d}{dt}(l\cos\theta)\right)^2$$

Упрощение выражения v_2 приводит к: $v_2^2 = \dot{x}^2 - 2l\dot{x}\dot{\theta}\cos\theta + l^2\dot{\theta}^2$. Лагранжиан теперь определяется по формуле:

$$L = \frac{1}{2}(M+m)\dot{x}^2 - ml\dot{x}\dot{\theta}\cos\theta + \frac{1}{2}ml^2\dot{\theta}^2 - mgl\cos\theta \tag{2.28}$$

и уравнения движения:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = F, \tag{2.29}$$
$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0.$$

Подстановка L в эти выражения с последующим упрощением приводит к уравнениям, описывающим движение обратного маятника:

$$(M+m)\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta = F, \tag{2.30}$$
$$l\ddot{\theta} - g\sin\theta = \ddot{x}\cos\theta.$$

Эти уравнения являются нелинейными, но, поскольку цель системы управления – удерживать маятник вертикально, то уравнения можно линеаризовать, приняв $\theta \approx 0$.

В некоторых дифференцированиях центростремительная сила ускорения, обусловленная угловой скоростью, не учитывается. Динамика в этой модели не исключает этого условия. Тем не менее, может быть безопасным пренебречь этим условием при проектировании системы управления из-за того, что сбалансированный маятник должен иметь низкую скорость.

Изначальные уравнения системы для дискретного решения на практике решаются с помощью переноса всех вторых производных в правую часть.

Затем каждое уравнение со вторыми производными становится двумя уравнениями с первыми производными, после чего они решаются с помощью численной интеграции методом Рунге-Кутты.

Далее необходимо показать, как будет происходить управление обратным маятником на подвижной платформе в целях получения его устойчивого положения.

2.5 ПИД-регулятор

Для контроля над движением обратного маятника воспользуемся параметрами ПИД-регулятора. Пропорционально-интегрально-дифференцирующий (ПИД) регулятор – устройство в управляющем контуре с обратной связью. Используется в системах автоматического управления для формирования управляющего сигнала с целью получения необходимых точности и качества переходного процесса. ПИД-регулятор формирует управляющий сигнал, являющийся суммой трёх слагаемых, первое из которых пропорционально разности входного сигнала и сигнала обратной связи (сигнал рассогласования), второе — интеграл сигнала рассогласования, третье — производная сигнала рассогласования. Можно изменять коэффициенты пропорционального, интегрального и диффиренциального контроля, чтобы проверить, как работает каждый компонент [7].

Пропорциональная составляющая вырабатывает выходной сигнал, противодействующий отклонению регулируемой величины от заданного значения, наблюдаемому в данный момент времени. Он тем больше, чем больше это отклонение. Если входной сигнал равен заданному значению, то выходной равен нулю [17].

Чем больше коэффициент пропорциональности между входным и выходным сигналом (коэффициент усиления), тем меньше статическая ошибка, однако при слишком большом коэффициенте усиления при наличии задержек

(запаздывания) в системе могут начаться автоколебания, а при дальнейшем увеличении коэффициента система может потерять устойчивость.

Интегрирующая составляющая пропорциональна интегралу по времени от отклонения регулируемой величины. Её используют для устранения статической ошибки. Она позволяет регулятору со временем учесть статическую ошибку [17].

Если система не испытывает внешних возмущений, то через некоторое время регулируемая величина стабилизируется на заданном значении, сигнал пропорциональной составляющей будет равен нулю, а выходной сигнал будет полностью обеспечиваться интегрирующей составляющей. Тем не менее, интегрирующая составляющая также может приводить к автоколебаниям при неправильном выборе её коэффициента.

Дифференцирующая составляющая пропорциональна темпу изменения отклонения регулируемой величины и предназначена для противодействия отклонениям от целевого значения, которые прогнозируются в будущем. Отклонения могут быть вызваны внешними возмущениями или запаздыванием воздействия регулятора на систему [17].

Балансировка перевернутого маятника является типичным примером при демонстрации системы управления. Вес на плече над поворотным шарниром является неустойчивой системой. У неё есть одна устойчивая точка, когда вес отлично сбалансирован. Если вес немного по обе стороны от этой точки, то система начнет отходить от этой устойчивой точки. С другой стороны, вес на плече ниже оси поворота является стабильной системой.

Система управления, используемая в имитации, является базовой линеаризованной системой управления. Используя приближение малого угла $sin\theta \approx 0$, система управления может быть упрощена. В то время как приближение верно для углов близких к нулю, при больших значениях аппроксимация ухудшается (см. таблицу 2.1).

Заметим, что $\theta = 0$, когда маятник находится в вертикальном положении. Затем система управления становится:

$$F = -k_P \cdot \theta - k_D \cdot \omega - k_I \cdot \int \theta dt, \quad (2.31)$$

где θ - угол;

k_P – пропорциональная составляющая ошибки угла;

k_D – дифференцирующая составляющая ошибки угла;

k_I – интегрирующая составляющая ошибки угла.

Прирост значений может быть изменен в зависимости от свойств, таких как масса и длина маятника.

Систематический способ итеративного улучшения работы - попробовать пару значений и наблюдать, есть ли перерегулирование и сколько. Превышение - это перекоррекция ошибки - или амплитуда «колебания» значений поля.

Также обратите внимание на время необходимое системе для достижения ее устойчивого состояния. Время установления стабильности - это время, которое требуется для остановки изменений значений поля.

Таблица 2.1 - Аппроксимация небольшого угла

Угол	Ошибка
15°	1.1%
30°	4.5%
60°	17.3%
90°	36.3%
120°	58.6%
150°	80.9%
180°	100.0%

Если много перерегулирований, можно уменьшить k_P или увеличить k_D. Если время успокаивания длительное, можно увеличить k_P. k_I используется для устранения установившейся ошибки в том случае, если маятник балансирован в течение длительного периода в положении, отличном от вертикального.

2.6 Подход к разработке ИТ-инфраструктуры виртуальной лаборатории

Инфраструктура – комплекс взаимосвязанных обслуживающих структур или объектов, составляющих и обеспечивающих основу функционирования системы [29].

Информационная инфраструктура включает совокупность информационных центров, банков данных и знаний, систем связи и обеспечивает доступ потребителей к информационным ресурсам.

Инфраструктура информационных технологий (ИТ-инфраструктура) – это организационно-техническое объединение программных, вычислительных и телекоммуникационных средств, связей между ними и эксплуатационного персонала, обеспечивающее предоставление информационных, вычислительных и телекоммуникационных ресурсов, возможностей и услуг работникам, необходимых для осуществления профессиональной деятельности и решения соответствующих бизнес-задач [29].

Чтобы эффективно использовать подход к разработке, программные приложения должны удовлетворять ряду архитектурно значимых требований (таких как возможность развертывания, модифицируемость, проверяемость и контролируемость) [23]. Эти требования запрашивают высокого приоритета и не могут быть легко изменены.

Особенностью правильной ИТ-инфраструктуры является то, что она начинается на уровне разработки кода. Поэтому при использовании инструментов для настройки нашей среды используем подход «инфраструктура как код». Инфраструктура больше не создается вручную, а является результатом автоматизированного процесса.

Построение инфраструктуры на уровне кода ведет к следующему виду результатов, к которым стремятся многие команды и организации. Подход «инфраструктура как код» дает следующие преимущества [29]:

– ИТ-инфраструктура поддерживает и позволяет изменять, а не является препятствием или ограничением;

– изменения в системе являются обычными, без стресса для пользователей или ИТ-персонала;

– ИТ-персонал тратит свое время на ценные дела, которые затрагивают их способности, а не на рутинные, повторяющиеся задачи;

– пользователи могут получать, предоставлять и управлять ресурсами, в которых они нуждаются, без необходимости, чтобы ИТ-персонал делал это за них;

– команды могут легко и быстро восстанавливаться после сбоев, а не предполагать, что отказ может быть полностью предотвращен;

– усовершенствования производятся непрерывно, а не через дорогостоящие и рискованные проекты;

– решение проблем доказано путем их внедрения, тестирования и измерения, а не путем обсуждения их на совещаниях и документах.

Так же интересует непрерывная интеграция, как часть DevOps (англ. CI, англ. Continuous Integration) – это практика разработки программного обеспечения, которая заключается в слиянии рабочих ответвлений проекта в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем [23].

Существует несколько типов динамических платформ инфраструктуры. Варианты для платформ для построения динамической инфраструктуры растут и меняются, так как все больше поставщиков предлагают решения. Следующая платформа является отправной точкой, чтобы разрабатывать подход. Она свободно основывается на определении развертывания моделей облака в определении облака Национального института стандартов и технологий, о чем упоминалось в разделе 1.2.

В данном случае положим за модель развертывания частное облако, предоставляющее программное приложение как услугу для пользователей.

Частное облако построено и работает для нескольких потребителей внутри одной организации [30]. Например, для отделов или команд внутри

компании. Поставщик услуг может разрабатывать, работать, или даже разместить частное облако для организации, но ресурсы облака не используются совместно с другими организациями.

Компания, как правило, платит за общую доступную память, даже если не она не вся используется, хотя компания может иметь внутренние траты или ограничение мощности для отделов на основе их использования и бюджетов.

Покажем справочную модель частного облака, которая может служить общим представлением той архитектуры, что в итоге должна получиться. (см. рис. 2.8)

Инфраструктура как сервис – это применение принципов частной облачной архитектуры для предоставления облачной инфраструктуры.

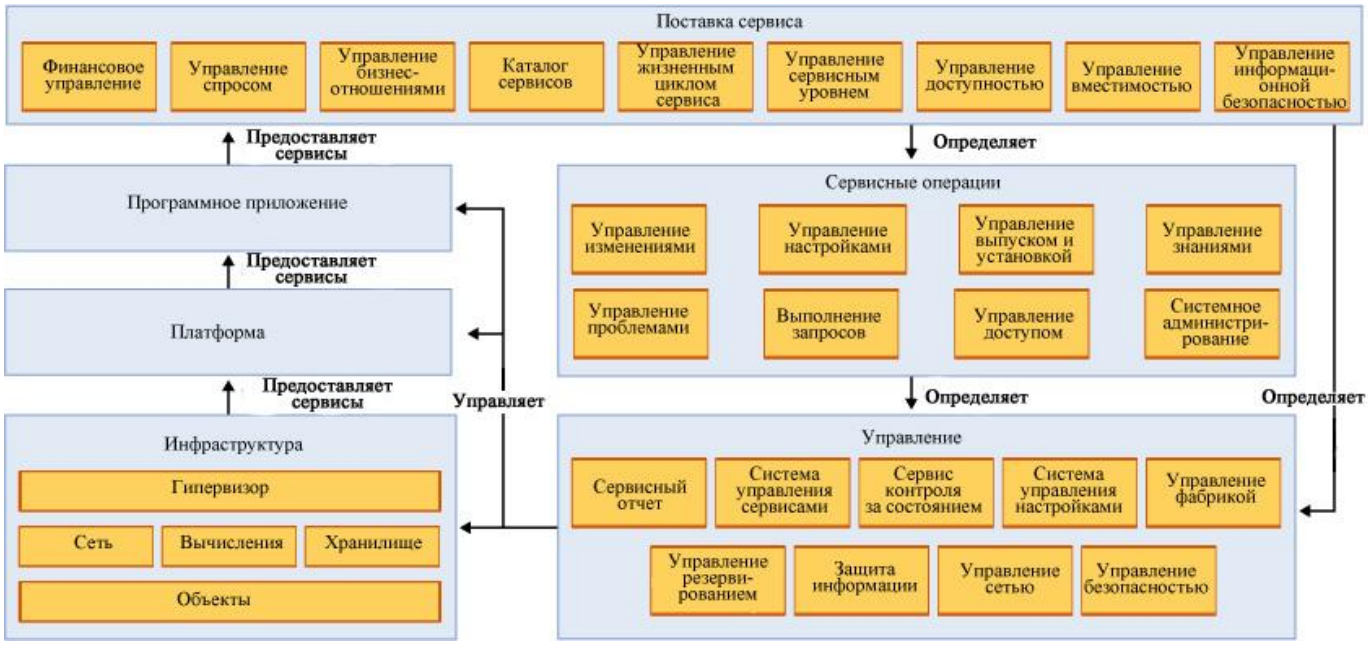

Рис. 2.8 – Справочный вид архитектуры частного облака с точки зрения модели «инфраструктура как сервис»

Можно использовать справочную модель, показанную на рисунке 2.8, чтобы гарантировать, что идет разработка комплексного решения, охватывающего все слои, необходимые для зрелого решения инфраструктуры как сервиса.

Модель выступает в роли «направляющих рельсов», чтобы помочь разработчикам и архитекторам облачной инфраструктуры в комплексном решении разработки частной облачной архитектуры. Эта модель используется только в качестве справочного материала, а некоторые части больше подчеркиваются в технической справочной архитектуре, основанной на опыте эксплуатации частных облаков в реальных средах.

Частное облако имеет несколько уровней организации. Каждый из них отвечает за свою функцию и выполняется соответствующих программным обеспечением. В таблице 2.2 можно увидеть архитектуру частного облака с точки зрения принципа инфраструктура как сервис.

Таблица 2.2 – Архитектура частного облака

Часть архитектуры	Уровни			
IaaS	Пользовательский и администраторский интерфейс			
Saas	SaaS нагрузка			
PaaS	PaaS нагрузка			
IaaS	Уровень оркестрации			
IaaS	Уровень управления			
IaaS	Уровень автоматизации			
IaaS	Уровень виртуализации			
IaaS	Уровень оборудования			
IaaS	Хранилище	Сеть	Вычисления	Объект

Уровень оборудования или уровень аппаратного обеспечения включает в себя объекты центра обработки данных и механические системы, а также систему хранения, сети и вычислительную инфраструктуру. Каждый из этих элементов должен обеспечивать взаимодействие интерфейсов управления с более высокими уровнями архитектуры. К примерам относятся серверы,

поддерживающие управление веб-службами (WS-Management) и массивы хранения, предоставляющие интерфейсы Windows PowerShell или Storage Initiative-Specification (SMI-S).

Таблица 2.3 – Краткая характеристика слоев архитектуры частного облака

Уровень	Характеристика	Пример ПО
Оборудования	Позволяет управлять и обновлять оборудование	Hyper-V Cloud Fast Start «Cube»
Виртуализации	Виртуализирует сервера, сети, хранилища в пул ресурсов	Virtual box, WindowsServer 2008 K2 Hyper-V
Автоматизации	Обеспечивает централизованную автоматизацию настроек сценариев	PowerShell 2.0
Управления	Группирует автоматизированные сценарии в процессы и операции	Менеджер виртуальной машины, менеджер защиты данных
Оркестрации	Группирует процессы и операции	Opails, Docker
Интерфейсы пользователя и администратора	Пользовательский интерфейс используется для запуска рабочих процессов	Виртуальный менеджер

Например, Virtual Box обеспечивают уровень виртуализации. Это позволяет нам использовать виртуальные машины (VM) и сети с VLAN, а также обеспечивать хранение через общие тома кластера и виртуальные диски. Уровень виртуализации помогает достичь нескольких основных характеристик Национального института стандартов и технологий, таких как объединение ресурсов и эластичность. Благодаря виртуализации можно быстрее распределять ресурсы.

Уровень автоматизации – это следующий слой стека снизу вверх. Уровни автоматизации, управления и оркестровки строятся на основе подробных и самых широких с точки зрения автоматизации ИТ-процессов. Самый низкий уровень – уровень автоматизации включает такие технологии, как, например,

Windows PowerShell 2.0, инструментарий управления Windows (WMI) и WS-Management. В данном случае, они могут быть основополагающими технологиями, обеспечивающими интерфейс между системами управления высшего уровня и физическими и виртуальными ресурсами.

Уровень управления состоит из нескольких продуктов, которые используют технологии уровня автоматизации для выполнения таких задач управления, как проверка соответствия исправлений, развертывание исправлений и проверка установки. Уровень управления обеспечивает базовую автоматизацию процессов, но обычно ограничивается одним аспектом жизненного цикла управления сервером (таким как развертывание, исправление, мониторинг, резервное копирование и т. д.).

Уровень оркестрации обычно не рассматривается в традиционных ИТ-средах, но очень важен для предоставления атрибутов облачности. Уровень оркестрации связывает несколько продуктов, технологий и процессов, чтобы обеспечить сквозную автоматизацию ИТ-процессов. Хотя, например, System Center Configuration Manager может автоматизировать развертывание исправлений, его интеграция с системой управления услугами или дополнительными продуктами и решениями сторонних производителей требует, чтобы уровень оркестровки координировал сквозной процесс для нескольких продуктов.

Уровень оркестрации помогает создавать рабочие процессы или запускать сценарии, которые могут автоматизировать сложные задачи, такие как развертывание кластера, исправление узлов и подготовка виртуальных машин.

Определение Национального института стандартов и технологий атрибута самообслуживания по требованию пользователя является новой концепцией для многих ИТ-организаций. Речь идет прежде всего об устранении барьеров между потребностями пользователей в ИТ-ресурсах и доставкой этих ресурсов. Например, в некоторых организациях это может занять до шести

месяцев с момента запроса нового сервера, чтобы он был легко доступен. Ограничения процесса и технологии вызывают задержку.

Для возможности самообслуживания необходим новый интерфейс, который позволяет пользователям запрашивать сервисы. Это чаще всего проявляется в портале самообслуживания. Этот портал предоставит пользователям каталог услуг, из которого они могут запросить такие объекты, как новая виртуальная машина.

В данной эталонной архитектуре определяем как интерфейс самообслуживания для потребителей, так и централизованный интерфейс администратора для информационных технологий. Для пользовательского интерфейса предоставляет самообслуживающийся портал, а для пользовательских сценариев – например, набор инструментов Dynamic Datacenter Toolkit.

Интерфейс администратор предоставляет базу данных управления конфигурацией, а также надежное решение для управления изменениями. Все обычные операции в нашем решении происходят как запросы на изменение. Они запускают автоматизированные рабочие процессы, например в Opalis. Именно так обеспечивается надлежащее управление изменениями, предоставляя современную автоматизацию.

Таким образом, были разработаны математические модели движения частицы в поле конденсатора, двойного маятника и обратного маятника на тележке и подсчитана погрешность показаний. Подход к разработке ИТ-инфраструктуры виртуальной лаборатории основывается на свойствах и архитектуре частного облака. Данная теоретическая основа потребуется для реализации в следующей главе.

ГЛАВА 3 РЕАЛИЗАЦИЯ ИТ-ИНФРАСТРУКТУРЫ И МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ ДЛЯ ОБЛАЧНОЙ ВИРТУАЛЬНОЙ ЛАБОРАТОРИИ

3.1 Выбор и обоснование технологий

Основой для разработки и тестирования виртуальной лаборатории положим принцип DevOps (англ. development и operations – разработка операции) – это совокупность практических приемов, позволяющих активно взаимодействовать и интегрировать людей из направления разработки и информационно-технологического обслуживания [31]. Данный принцип основан на идее о тесной взаимозависимости разработки и эксплуатации программного обеспечения и использование его обосновано в рамках разработки облачных технологий [31].

Непосредственная черта данной методологии фокусируется на стандартизации окружений разработки с целью способствования быстрому выпуску релизов.

Задача DevOps – сделать процесс разработки и поставки программного обеспечения согласованным с эксплуатацией, часто эти задачи решаются при поддержке автоматических средств.

Описанный ниже стек технологий, выбранных для проекта, состоит только из свободно распространяемого программного обеспечения и соответствует принципам DevOps.

Прежде всего, используем Docker как программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Docker позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой групп в ядре, а также предоставляет среду по управлению контейнерами [28].

Особенностью установки Docker является то, что на Windows потребуется виртуальный образ Linux-подобной системы. В виду данной ситуации, воспользуемся интерфейсом командной строки – скриптовым языком Vagrant для создания удобного окружения. Такой кардинальный способ позволит изолировать приложение от остальной системы и ошибки в нем не скажутся на работе всей системы в целом. Так же преобладают следующие положительные стороны [46]:

- быстрый способ развернуть на сервере готовый проект ценой накладных расходов на виртуализацию;
- возможность проверить, как ведет себя распределенное приложение при сетевых проблемах и падении машин;
- для тестирования есть начальный образ системы, к которому можно откатиться перед следующим прогоном тестов;
- удобно мигрировать приложение с хоста на хост;
- в противовес deb- и rpm-пакетам виртуализация разрешать конфликты зависимостей приложений с ошибками, создавая для них изолированную среду.

По сравнению с аналогами (Chef или Puppet) получаем следующие преимущества:

- при внедрении непрерывной поставки новой версии проекта необходимо настроить несколько экземпляров нашей среды для каждой стадии конвейера поставки (этап фиксации, приемочный тест, тесты производительности, предварительная подготовка и производство). Использование физических машин для этой цели нецелесообразно;
- установка приложения и необходимой среды становится легкой, потому что всегда начинаем с пустой виртуальной машины. Старую виртуальную машину можно полностью удалить, а новая будет уже настроена;
- легкая виртуализация. Запуск контейнера Docker происходит намного быстрее, чем запуск виртуальной машины, поскольку гостевую операционную систему не требуется загружать. Это уменьшает накладные расходы. Несколько контейнеров совместно используют ядро операционной системы хоста, но

имеют свою собственную файловую систему, пользователей, сеть и процессы. С точки зрения операционной системы хоста мы просто запускаем другой процесс, когда запускаем контейнер Docker. Это значительно ускоряет запуск контейнера, обеспечивая при этом хорошую изоляцию контейнеров.

По этой причине Chef или Puppet часто используются в сочетании с виртуализацией: запускается виртуальная машина, а Chef или Puppet используются для настройки среды в виртуальной машине.

Надстройка Docker Compose - это инструмент для определения и запуска многоконтейнерных приложений Docker [24]. Docker Compose позволяет указать связь между несколькими контейнерами в yml-файле, а затем запустить все эти контейнеры одновременно. Синтаксис для команды docker-compose.yml довольно прост. Вы указываете имя контейнера, образа, который он должен использовать, и команду запуска для контейнера. Кроме того, вы можете указать отображение портов на порты хост-машины. Затем, используя одну команду, создаются и запускаются все службы из конфигурации. Это позволяет:

- иметь несколько изолированных сред на одном хосте;
- только обновить контейнеры, которые были изменены;
- переменные и перемещение композиции между средами.

Compose отлично подходит для разработки, тестирования и промежуточной среды, а также для рабочих процессов непрерывной интеграции.

Язык программирования серверной части выберем Java, так как есть опыт его использования, как и множество технологий для данного языка упрощающих разработку веб-приложений.

Фреймворк Spring включает в себя набор различных технологий и позволит использовать фреймворки аспектно-ориентированного программирования, доступ к данным, MVC, тестирования. Особенности фреймворка Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений

на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк [34].

Spring Boot позволяет легко создавать автономные модули класса Spring приложений, которые настроены так, что можно начать работу с минимальными усилиями [26]. Большинству приложений Spring Boot нужно очень мало конфигураций Spring.

Особенности технологии:

- создание автономных приложений Spring;
- встроенный напрямую Tomcat, Jetty или Undertow (нет необходимости развертывания WAR-файлов);
- обеспечение опциональных «стартовых» файлов POM для упрощения конфигурации;
- автоматическая настройка Spring всякий раз, когда это возможно;
- обеспечение производства готовых функций, таких как метрики, проверки работоспособности и внешние конфигурации;
- отсутствие генерации кода и нет необходимости заниматься конфигурацией XML.

Отдельно выделим дочерний фреймворк Spring Security, отвечающий за аутентификацию и авторизацию. Он позволяет защитить Java-приложений от различных типов хакерных атак и позволяет интегрирование с Spring Web MVC.

Воспользуемся Hibernate как библиотекой для языка программирования Java, предназначенной для решения задач объектно-реляционного отображения [25].

jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Данная библиотека поможет отделить поведение интерфейса от его html-структуры.

Виртуальные физические модели будут разработаны как JSON-файл, позволяющие загрузить все параметры и отрисовать модель с помощью библиотек JavaScript.

На стороне клиента-браузера потребуется интерфейс. HTML, CSS, JS библиотека Bootstrap делает веб-фронтенд разработку быстрее и проще. Она сделана для людей разных уровней квалификации, устройств всех форм и проектов любого масштаба.

Базу данных возьмем PostgreSQL, так как она является объектно-реляционной системой управления базами данных на основе POSTGRES, версия 9.5. PostgreSQL поддерживает большую часть стандарта SQL и предлагает множество современных функций, необходимых в приложении:

- сложные запросы;
- внешние ключи;
- триггеры;
- представление;
- транзакционная целостность;
- управление конкурентным доступом с помощью многоверсионности;
- формат JSON.

Для подключения зависимостей и библиотек в Java-код используем gradle, в нем понятная логика сборки, хорошая документированность, например, в отличие от Maven, есть проверка результатов и результатов сборки (такие как результат сборки, стандартный вывод / ошибка, выполненных заданий и т. д.), межверсионное тестирование на совместимость, отладка сборки при тестировании из IDE [22].

Потребуются отдельные разветвления проекта, где будут разрабатываться разные модули. Воспользуемся распределённой системой управления версиями git для контроля над всеми разветвлениями проекта, которая позволит вернуться к предыдущей рабочей версии в случае ошибок и создавать сборку последней рабочей версии, и является необходимой частью DevOps.

Перейдем к разработке инфраструктуры виртуальной лаборатории на основе выбранных средств реализации.

3.2 Разработка инфраструктуры виртуальной лаборатории

После обоснования необходимости и полезности средств разработки компонентов виртуальной лаборатории следует перейти к ИТ-инфраструктуре лаборатории, которая определит способ её использования и расширения.

С учетом вышесказанного разработаем ИТ-инфраструктуру, которая будет соответствовать принципам DevOps и добьемся её оптимальности измеряемой в объеме занимаемой памяти и скорости установки с развертыванием.

Созданные исполняемые сценарии в файле gradle.build позволяют руководить сборкой проекта и копированием артефактов в отдельную папку, артефактами являются схема базы данных, скомпилированный jar-файл веб-приложения и Dockerfile для самой базы и веб-приложения, где указаны инструкции для установки контейнеров Docker. Gradle вызывает Vagrant.

Сценарий, описанный в Vagrantfile, позволяет вызвать программное средство виртуализации и развернуть объект с операционной системой, с Docker и его контейнерами, причем доступ к приложению можно получить по заданному в Vagrant файле IP-адресу и порту.

Docker является частью широкого подхода к вычислениям в пользу использования микросервисов. Микросервисы – небольшие, быстрые программные приложения, которые работают в автономных модулях [23]. Границы обслуживания каждого компонента упрощают создание модульных систем. Микросервисы строятся вокруг продуктов, а не проектов. Вместо того, чтобы рассматривать программное обеспечение как проект, который должен быть завершен и передан, продукт-подход увеличивает сотрудничество и собственность между разработчиками и заинтересованными сторонами. Кроме того, микросервисы разработаны с ошибками - это означает, что они создаются независимо от качества, чтобы система могла продолжать работать каждый раз,

когда отдельные компоненты выходят из строя. На рисунке 3.1 можно увидеть команды копирования Dockerfile в отдельную папку, остановку и создание виртуальной машины.

```
jar {
    baseName 'vpl'
}
task startVirtualBoxVm(type: VagrantUp) {
    description = 'Starts VM machine running on VirtualBox.'
    group = 'VirtualBox VM'
    boxDir = virtualBoxDir
}

task stopVirtualBoxVm(type: VagrantDestroy) {
    description = 'Stops VM machine running on VirtualBox.'
    group = 'VirtualBox VM'
    boxDir = virtualBoxDir
}
task copyDockerfile(type: Copy) {
    from 'config/docker/app'
    into 'artifacts/config/docker/app'
}
task copyDockerfileDb(type: Copy) {
    from 'config/docker/db'
    into 'artifacts/config/docker/db'
}
```

Рисунок 3.1 – Часть сценария gradle.build

На рисунке 3.2 можно увидеть основные возможности использования образов Docker. Docker имеет несколько основных понятий.

Слой – набор файлов только для чтения, предназначенных для обеспечения системы. Можно представить слой как о снимок только для чтения файловой системы.

Образ – слой только для чтения, который является основой контейнера. Он может иметь родительский образ, чтобы абстрагироваться от базового моментального снимка файловой системы. Таким образом, образ Java наследуется от образа Linux с предустановленными утилитами. Образ tomcat будет иметь образ Java в качестве родителя, потому что он зависит от Java для запуска Tomcat.

Контейнер – выполняемый экземпляр образа, в основном это процесс, изолированный Docker, который работает поверх файловой системы, предоставляемый образом.

Рисунок 3.2 – Возможности использования Docker-образа

Реестр/концентратор является центральным местом, где «живут» все публично опубликованные образы. Вы можете искать его, загружать свои изображения туда, и когда вы тянете изображение докера, он приходит в репозиторий/концентратор.

Docker-машина – это виртуальная машина, в которой можно запускать контейнеры Docker. В Linux можно запускать Docker-контейнеры изначально, но в OSX и Windows нужен слой абстракции. Docker-машина будет использовать легкую виртуальную машину, которая хорошо интегрируется с утилитами командной строки Docker.

Docker compose - это утилита для запуска нескольких контейнеров как системы. Она будет заботиться о том, чтобы они знали друг о друге и удостоверились, что они правильно связаны друг с другом. Это означает, что можно запускать приложение в одном контейнере, а база данных - в другом контейнере, а приложение аналитики - в другом контейнере и так далее. Это конечная изоляция, и это означает, что приложения независимы и выполняются

в процессе разработки очень похоже на то, как система может работать в процессе производства.

Основное отличие образов Docker от других виртуальных машин заключается в том, что образы Docker делят ядро Linux с хост-машиной. Для конечного пользователя основным следствием этого является то, что любой Docker-образ должен быть основан на Linux-системе с Linux-совместимым программным обеспечением, которое включает в себя (Python, Matlab и большинство других научных потребностей программирования). Разделение ядра Linux делает Docker гораздо легче и производительнее, чем полные виртуальные машины [37].

Развернем среду приложения с помощью файла Docker, чтобы его можно было воспроизводить в любом месте. Важной частью использования Docker – это минимизации объема памяти контейнеров и их образов. Меньшие образы имеют немало приемуществ. Помимо очевидного размера, минимизация делает среду небольшой и эффективной. Небольшие образы повышают уровень безопасности при уменьшении размера зоны безопасности [24]. В данном случае образы отвечают за сервис разворачиваемой базы данных, сервис управления базой данных Postgres, минимальный основной образ и за разработанное веб-приложение.

Определим сервисы, составляющие приложение в docker-compose.yml, чтобы они могли работать вместе в изолированной среде, в данном проекте указана директория, где лежат файлы для сборки с помощью Docker, и какие порты будут занимать созданные контейнеры.

На первом этапе оптимизации начнем с меньшего базового образа. Нет необходимости загружать все пакеты Ubuntu, поэтому начнем с меньшего базового изображения, такого как Alpine, которое, вероятно, станет базовым для всех официальных образов Докера (Jenkins, Maven) [28]. Это базовое изображение весит около 5 МБ, тогда как Ubuntu - около 188 МБ. На рисунке 3.3, 3.4 можно увидеть сравнение базовых образов Docker.

Благодаря однослойной структуре данный образ займет минимальное количество памяти, также его использование надежно [32].

Проведем ряд усовершенствований в написании сценария Dockerfile. Контейнер, созданный с помощью образа, который определяет Dockerfile, должен быть как можно более эфемерным.

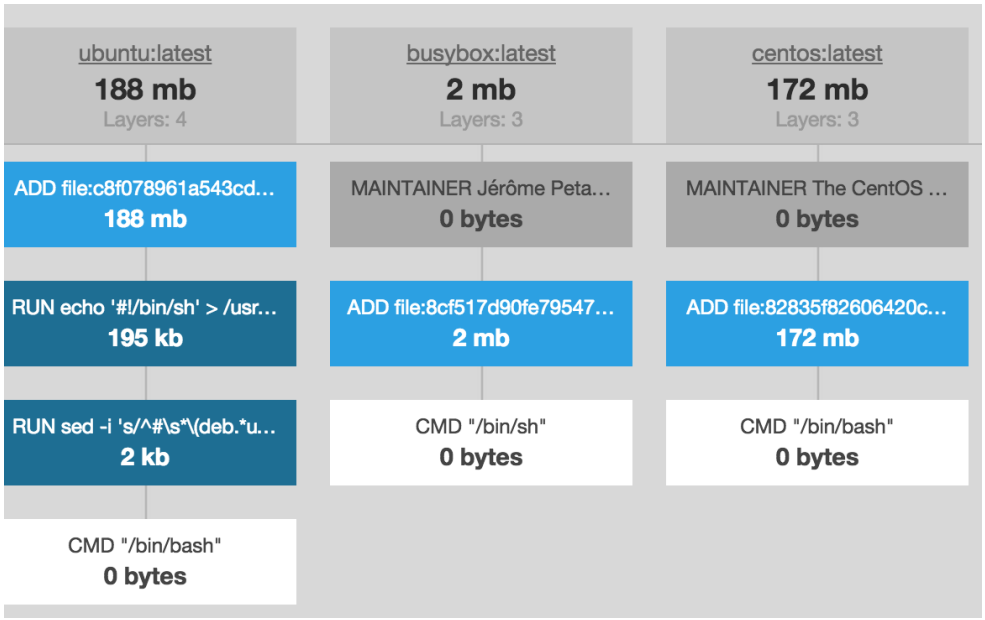

Рисунок 3.3 – Сравнение базовых образов с тремя и четырьмя слоями

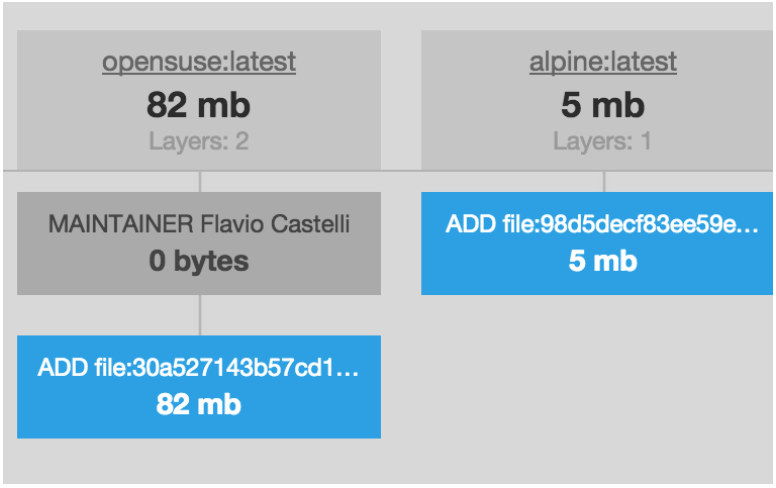

Рисунок 3.4 – Сравнение базовых образов с двумя и один слоем

Под «эфемерным» подразумевается, что его можно остановить и уничтожить, а новый построить и ввести в действие с абсолютным минимумом настроек и конфигурации.

В большинстве случаев лучше поместить каждый файл Dockerfile в пустой каталог. Затем добавить в этот каталог только файлы, необходимые для создания Dockerfile. Чтобы увеличить производительность сборки, можно исключить файлы и каталоги, добавив в нее файл .dockerignore. Этот файл поддерживает шаблоны исключений, похожие на файлы .gitignore.

На втором этапе, чтобы уменьшить сложность, зависимости, размеры файлов и время сборки, учтем, что нужно избегать установки дополнительных или ненужных пакетов только потому, что они могут пригодиться.

В процессе создания изображения Docker выполнит инструкции в созданном файле Dockerfile, выполнив каждую из них в указанном порядке. По мере изучения каждой команды Docker будет искать существующее образы в своем кэше, которое он может повторно использовать, а не создавать новый (дублирующий) образ. Если не нужно использовать кэш вообще, можно использовать опцию --no-cache = true в команде построения Docker. Однако, если не позволять Docker использовать свой кэш, то очень важно понять, когда он найдет подходящий образ или не найдет его. Основные правила, которым нужно следовать Docker, изложены ниже:

- начиная с базового образа, который уже находится в кэше, следующая команда сравнивается со всеми дочерними образами, полученными из этого базового образа, чтобы увидеть, была ли одна из них построена с использованием той же самой команды. Если нет, кэш становится недействительным;
- в большинстве случаев достаточно просто сравнить инструкцию в файле Dockerfile с одним из дочерних образов. Однако некоторые инструкции требуют дополнительного изучения и объяснения. Для команд ADD и COPY содержимое файла (файлов) на образе исследуется, и для каждого файла рассчитывается контрольная сумма. Время последнего изменения и последнего

доступа к файлу(-ам) в этих контрольных суммах не учитывается. Во время поиска в кэше контрольная сумма сравнивается с контрольной суммой в существующих образах. Если что-то изменилось в файле(-ах),например содержимое и метаданные, тогда кэш становится недействительным;

- помимо команд ADD и COPY, проверка кэша не будет рассматривать файлы в контейнере для определения совпадения с кэшем. Например, при обработке команды обновления RUN apt-get -y файлы, обновленные в контейнере, не будут проверяться, чтобы определить, существует ли кэш. В этом случае для нахождения совпадения будет использоваться только сама командная строка. После того как кэш становится недействительным, все последующие команды Dockerfile будут генерировать новые образы, и кэш не будет использоваться.

На рисунке 3.5 UML диаграмма последовательности развертывания инфраструктуры виртуальной лаборатории.

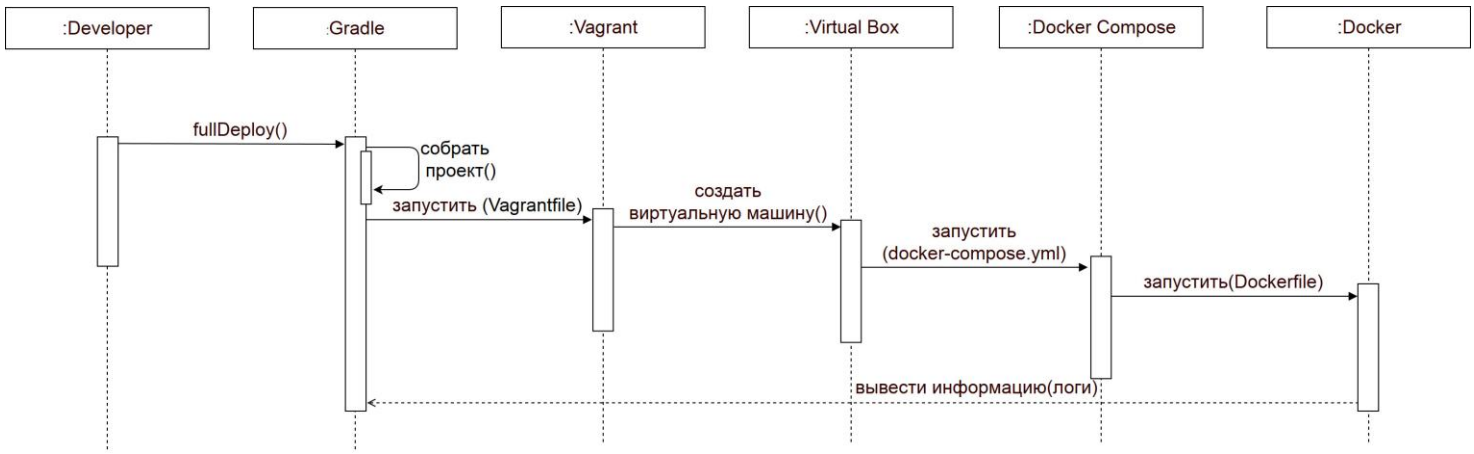

Рисунок 3.5 – Диаграмма последовательности развертывания инфраструктуры виртуальной лаборатории

Также важен способ написания в Dockerfile инструкции RUN. Для того чтобы сделать Dockerfile более читаемым, понятным и поддерживаемым, нужно разбить длинные или сложные инструкции RUN на несколько строк, разделенных обратными косыми чертами.

Вероятно, наиболее распространенным вариантом использования для RUN является команда apt-get, потому что она устанавливает пакеты, как раз она может имеет несколько ошибок, которые нужно учесть.

На третьем этапе, нужно избегать RUN apt-get upgrade или dist-upgrade, так как многие «существенные» пакеты из базовых образов не будут обновляться внутри непривилегированного контейнера. Если пакет, содержащийся в базовом образе, устарел, нужно обратиться к тому образу, что его поддерживает. Если известно, что есть определенный пакет, например foo, который необходимо обновить, нужно использовать apt-get install -y foo для автоматического обновления.

На четвертом этапе, нужно найти баланс между читаемостью (и, следовательно, долговременной поддерживаемостью) Dockerfile и минимизацией количества используемых слоев. На рисунке 3.6 можно посмотреть, как менялась занимаемая память на разных этапах оптимизации Docker-образов.

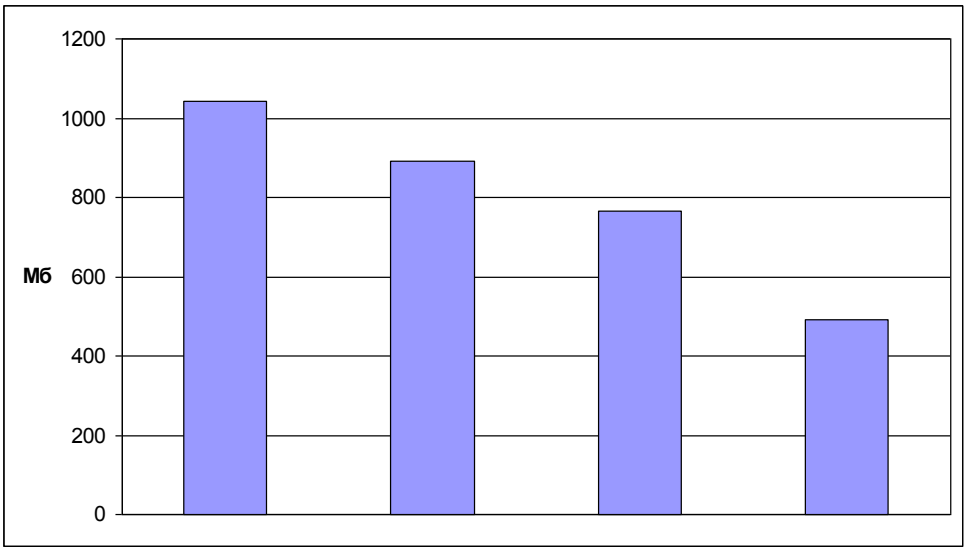

Рисунок 3.6 – Изменения объема занимаемой памяти

Хотя ADD и COPY функционально подобны, вообще говоря, команда COPY является предпочтительнее. Это потому, что она более прозрачная, чем ADD. COPY поддерживает только базовое копирование локальных файлов в

контейнер, в то время как ADD имеет некоторые функции (например, извлечение tar-файлов локально и поддержку удаленного URL-адреса), которые не сразу очевидны. Следовательно, ADD лучше всего использовать автоматическое извлечение локального файла tar в образ, как в ADD rootfs.tar.xz/.

Данный подход к разработке инфраструктуры может быть использован для любого проекта, основанного на облачных технологиях и представляющего части системы как отдельные сервисы.

3.3 Разработка инфраструктуры для работы с виртуальными моделями

Использование и редактирование виртуальной модели в облачной виртуальной лаборатории подразумевает необходимость обобщенного подхода к работе с объектом, тип которой имитационная модель.

Особой частью виртуальных моделей является используемый движок. От него зависит разработка точности моделей, имитирующих физические процессы. Есть две различные группы доступных физических движков, один из них «научный движок», другой – «движок общего назначения» [42].

Научные движки ориентированы на крайне точные расчеты, делающие моделирование как можно ближе к реальности. Тем не менее, подобная точность отражается на стоимости, эти физические движки требуют много вычислительной мощности и использование больших суперкомпьютеров, чтобы запустить виртуальные модели. Поскольку в данной работе сконцентрируется внимание на доступных компьютерах, которые позволяют осуществлять моделирование в аудиториях университетов, школ, а также на мобильных устройствах, имеющих низкую тактовую частоту процессора, подобные физические движки не подходят.

Другой доступный физический движок является «движком общего назначения». Этот тип движков упрощает моделирование, избегая таких вещей, как, например, броуновское движение, и поэтому экономит много

процессорного времени, следовательно, может работать на менее мощных машинах. Подобный тип физического движка подходит для проекта обучающей виртуальной физической лаборатории.

Разработаем структуру предметно-ориентированного языка движка. Предметно-ориентированный язык (англ. Domain-specific language, DSL) — язык программирования [41], специализированный для конкретной области применения. Построение такого языка и его структура данных отражают специфику решаемых с его помощью задач.

Для реализации выберем стандарт типа JavaScript Object Notation (нотация объектов JavaScript, JSON). Благодаря нему есть ещё одна альтернатива в выборе форматов.

Данный формат был выбран, так как нас интересует разбор файла на стороне клиента с помощью JavaScript, а данный формат является преимуществом, потому что он родной для JavaScript, JSON также позволяет формировать более сложные структуры, чем простые пары «имя/значение». Например, в отличие от xml, можно представлять массивы и сложные объекты, а не только простые списки ключей и значений [45]. Метод JSON.parse сериализует данные из полученного объекта согласно своей спецификации.

Для обеспечения возможности вычислять произвольные функции будет использоваться метод Java Script eval(code), который позволяет выполнить вычисления, переданные ему в виде строки.

Для того, чтобы позволять использовать несколько функций с разной областью определения, будем использовать список в блоке formulas, а не просто объект. Массив «finish-conditions», содержащий в себе границы значений параметров, массив «parameters», отвечающий за все используемые параметры и объект «graphics», описывающий графические элементы, используемые для анимации.

Базовая структура JSON-файла модели представлена на рисунке 3.7. Подробнее опишем принцип разборки полученного JSON-файла на примере модели движения частицы в обкладках конденсатора.

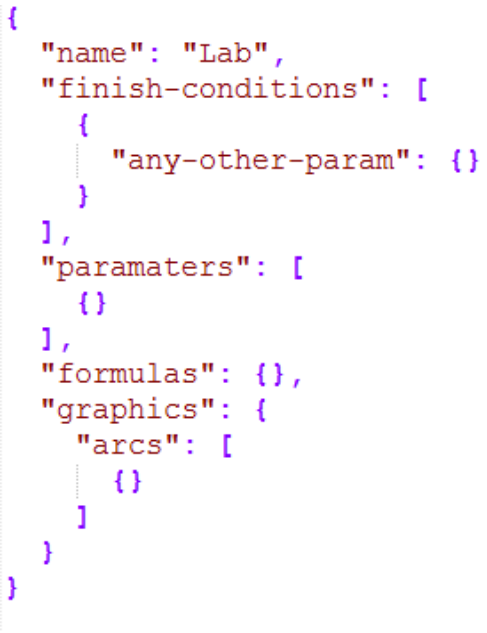

Рис. 3.7 – Базовая структура JSON-файла виртуальной модели

Сначала рисуется координатная плоскость, название, параметры берутся из блока graphics, заметем рисуется частица в позиции x_0, y_0.

Считываются параметры в словарь params. Считываются рекурсивно, то есть, если в initial-value есть используется другой параметр, то считывается этот параметр, и т.д. Для этого в выражении значений параметров используется ссылка на эту мапу, т.е. "initial-value": "params(\"param-name\")".

Для хранения параметров используется специальный объект, содержащий все нужные поля. Чтобы заметить бесконечную рекурсию (в случае ошибки в JSON), создадим список параметров, которые уже были запрошены. При попытке достать рекурсивно параметр будем проверять, нет ли его в том списке. Если есть - кидаем исключение. Подобным образом при разборе делается проверка на бесконечную рекурсию в поле formula (чтобы потом не проверять это в цикле). Параметры с editable=true доступны пользователю для редактирования в специальной форме; форма создаётся со значением из initial-value, а пользователь его меняет, если хочет. Параметры с displayed=false не выводятся на экран, с displayed=true - выводятся.

Считываются формулы для x и для y, каждая в свой массив. Формулы хранятся в виде объектов со всеми полями.

Считываются "finish-conditions". Каждое условие (например, см. рис. 3.8)

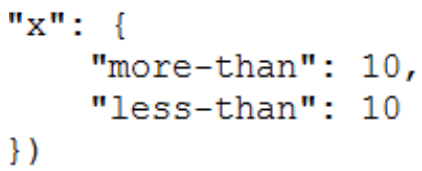

Рис. 3.8 – Пример параметров JSON-файла виртуальной модели

считывается в объект, содержаший поля more-than, less-than и т.п. Затем этот объект кладётся в словарь по ключу "x". Потом этот словарь кладётся в список условий завершения (так как их может быть несколько).

Далее следует алгоритм выполнения модели.

Начало итерации.

Вычисляются все параметры, у которых тип calculable. Делается это тоже рекурсивно. Для вычисления используется eval, в который подставляется значение поля formula. Значения параметров с displayed=true обновляются на экране.

По положению частицы выбирается формула с нужным диапазоном. Если не найдено, можно кинуть исключение, а можно завершить выполнение, как хочешь. Если найдено больше одной - исключение, либо берём первую найденную, тоже на твоё усмотрение.

Формулы выполняются с помощью eval, находятся значения x и y, частица перерисовывается в новой позиции.

Конец итерации.

Итерации проходят в цикле с предусловием: while(!areFinishConditionsSatisfied()). В условии пробегаемся по списку finish-conditions. Хоть одно выполняется - прекращаем цикл.

При проверке одного условия (которое, как мы помним, хранится в виде словаря) проходим по набору ключей словаря, хранящей это условие. Для

ключей x, y и t на соответствие условию проверяются эти переменные, для всех остальных ключей значения берутся из словаря params. В общем виде алгоритм считывания параметров представлен на рисунке 3.10.

Модель в виде JSON-файла загружается администратором в виртуальную лабораторию посредством интерфейса при создании новой лабораторной работы. В форме JSONB база данных сохраняет файл, который будет загружаться при открытии соответствующей страницы с виртуальной лабораторной работой.

Файл принимается с помощью асинхронного GET запроса jQuery.ajax(), который обращается по методу интерфейса программного приложения getJsonModel(int labId) с идентификатором необходимой виртуальной модели.

Интерфейс сайта можно увидеть на рисунке 3.9.

Благодаря Thymeleaf, который предлагает набор интеграций Spring, позволяющих использовать его как полнофункциональную замену JSP в приложениях Spring MVC, сериализованные объекты встраиваются на страницу.

Интегрированные Thymeleaf-обекты имеет немало преимуществ.

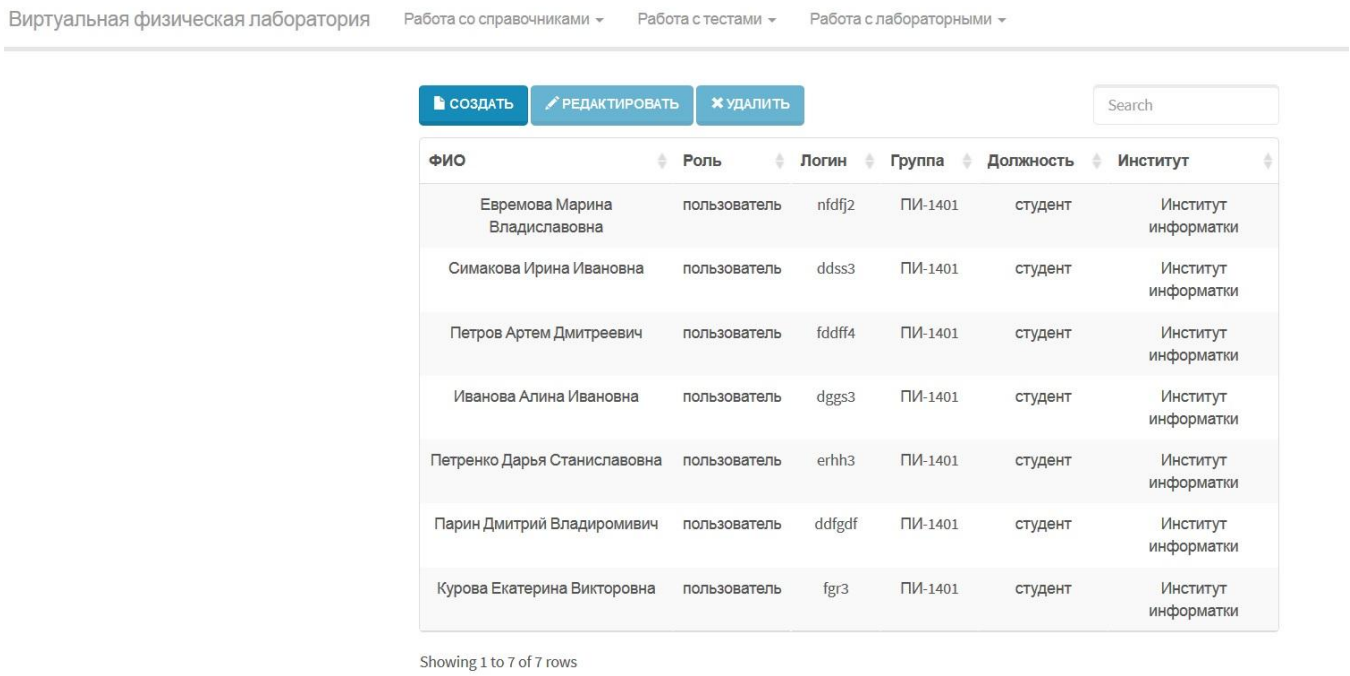

Рис. 3.9 – Интерфейс виртуальной лаборатории для администратора

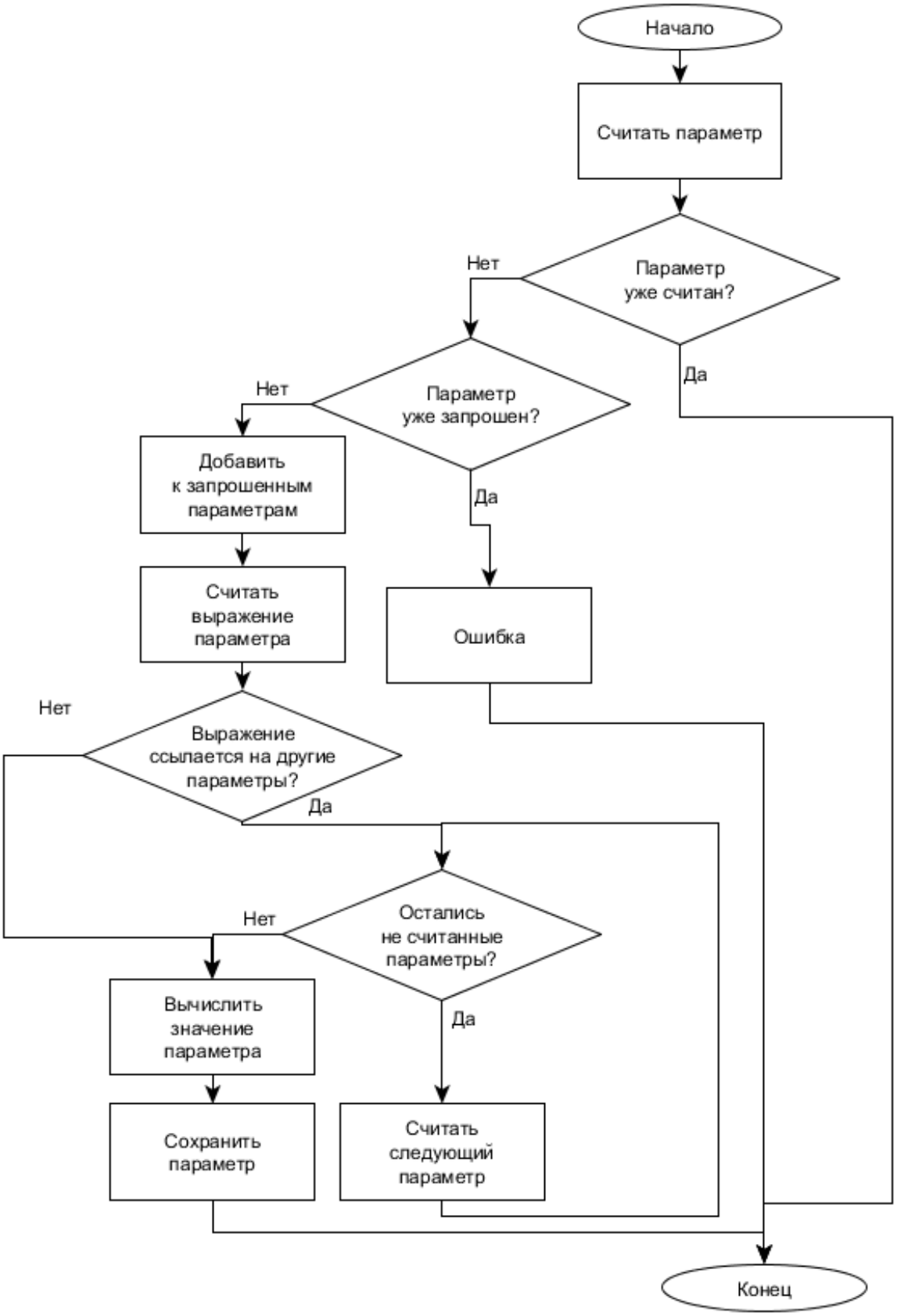

Рис. 3.10 – Алгоритм считывания параметров JSON-файла

Эти интеграции позволяют [34]:

- изменить методы в объектах Spring MVC @Controller на шаблоны, управляемые Thymeleaf, точно так же как в JSP;
- использовать Spring Expression в своих html template-страницах;
- создавать формы в шаблонах, которые полностью интегрированы с бинам серверной части и привязками к результатам, включая использование редакторов свойств, служб преобразования и обработки ошибок проверки, отображать сообщения об интернационализации из файлов сообщений, управляемых Spring (через обычные объекты MessageSource).

Thymeleaf предлагает режим «сценариев» для своих встроенных возможностей, так что можно интегрировать свои данные в сценарии, созданные на других языках и с помощью разметки, например <script th:inline="javascript">, добавлять на страницу.

3.4 Виртуальные модели

Покажем разработанные виртуальные модели физических явлений.

Движение частицы отслеживается с помощью линии. На рисунке 3.11 можно заметить, что напряженность за пределами обкладок стремительно убывает.

Частица в конденсаторе стартует с позиции (x0, y0) и заканчивает двигаться, когда выполняется одно из условий в finish-conditions. В коде тип словаря с параметрами params отвечает за все параметры, кроме x, y и t. Те, что имеют тип calculable, вычисляются на каждом шаге; они могут зависеть друг от друга и это тоже нужно учитывается коде. Формул для x и y может быть несколько, каждая определена на своём отрезке t.

На рисунке 3.12 наглядно моделируется система из диполя, которая имитирует краевые эффекты обкладок конденсатора. Также среди вариантов данной модели присутствует реализация, где краевые эффекты отсутствуют и заменяются ускорением свободного падения, которое действует на заряженную частицу.

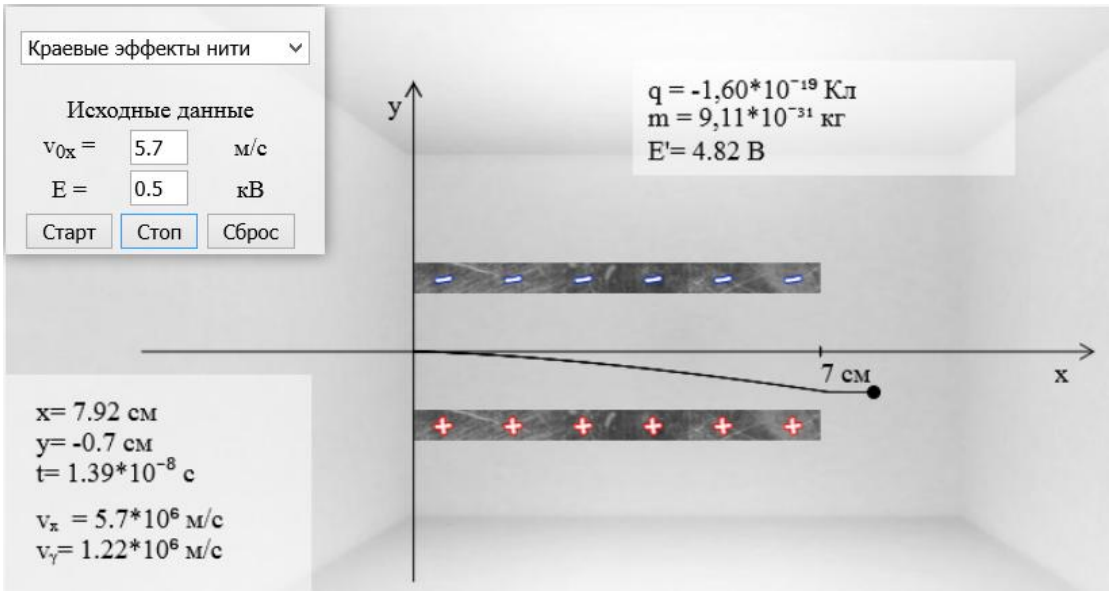

Рис. 3.11 – Модель движения электрона в обкладках конденсатора с краевым эффектом

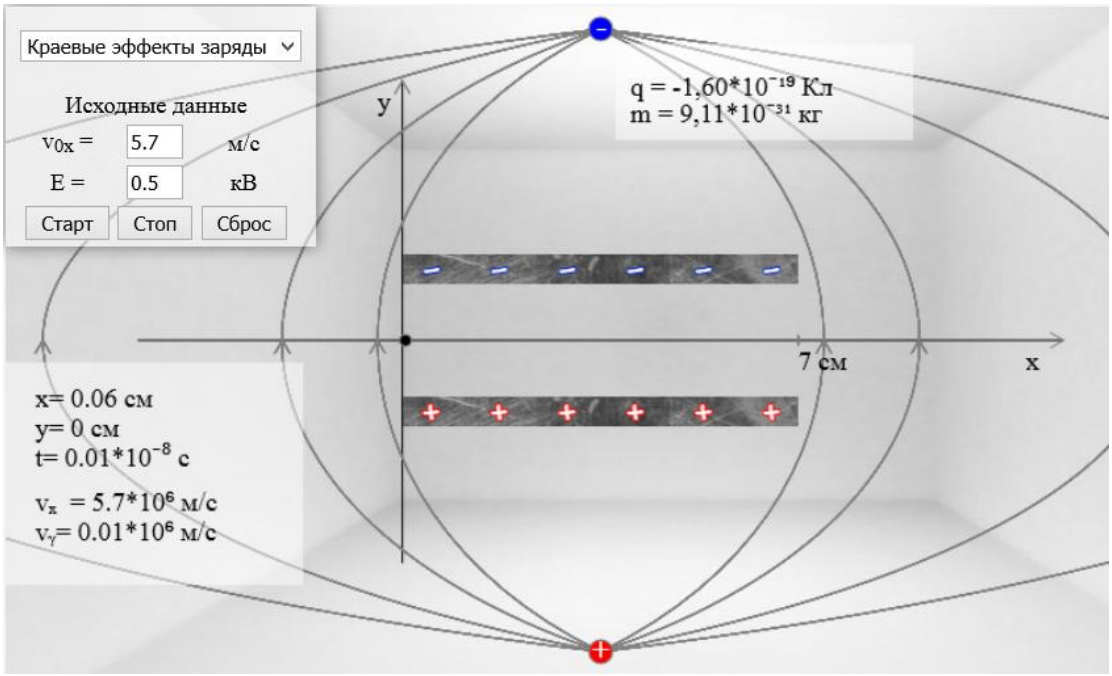

Рис. 3.12 – Модель движения электрона в обкладках конденсатора с диполем

Для облегчения работы в случае двойного маятника была использована библиотека Ember.js. Объекты JavaScript не поддерживают наблюдение за изменениями значений свойств. Следовательно, если какой-либо объект

собирается участвовать в Ember, вы можете увидеть объект Ember.Object вместо обычного объекта.

Ember.Object также предоставляет систему классов, поддерживающую такие функции, как mixins и методы конструктора [29]. Некоторые функции объектной модели Ember отсутствуют в классах JavaScript или общих шаблонах, но все они максимально приближены к языку и предлагаемым дополнениям.

Ember также расширяет прототип JavaScript Array с его интерфейсом Ember.Enumerable, чтобы обеспечить наблюдение за изменениями для массивов. Наконец, Ember расширяет прототип String с помощью нескольких методов форматирования и локализации. В данном случае для параметров маятника необходимо использовать Ember.Component. На рисунке 3.13 можно увидеть виртуальную модель двойного маятника.

Handlebars.js добавляет несколько дополнительных функций, чтобы упростить написание шаблонов, а также вносит небольшие детали, после чего частично меняется работа [43]:

- хелперы;
- выражения блоков;
- буквенные значения;
- разграниченные комментарии.

Блочные выражения имеют тот же синтаксис, что и секции Mustache, но не следует путать их друг с другом. Разделы похожи на неявные операторы each или with, зависящие от входных данных, а хелперы являются явными фрагментами кода, которые могут свободно реализовывать любое поведение, которое им нравится. Спецификация Mustache определяет точное поведение секций. В случае конфликтов имен хелперы получают приоритет.

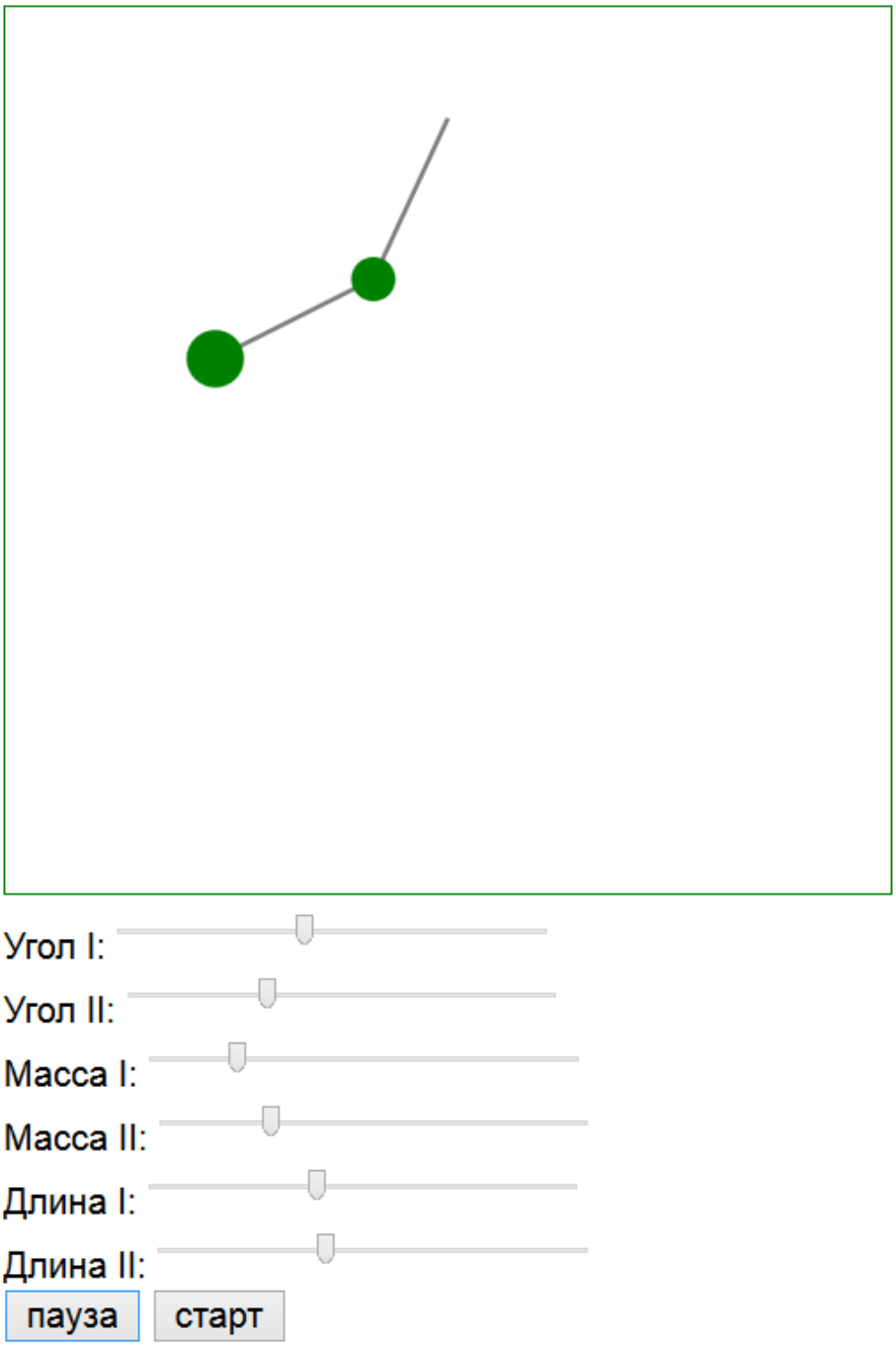

Рис. 3.13 – Модель двойного маятника

В виртуальной модели, представляющей собой математическое моделирование обратного маятника на подвижной платформе, основными составляющими частями является сам маятник с указанным весом и подвижная платформа, также имеющая вес. На рисунке 3.14 можно увидеть, как выглядит данная система.

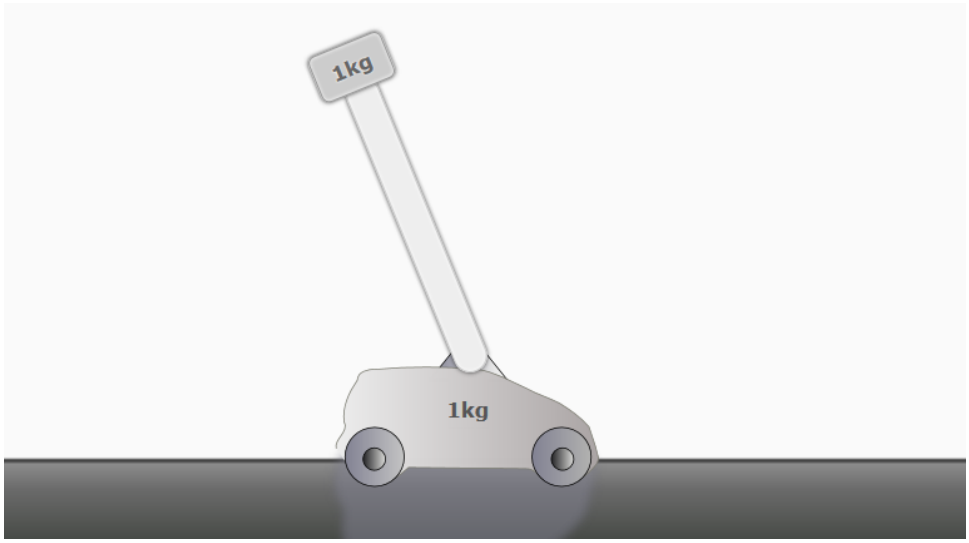

Рис. 3.14 – Модель обратного маятника на тележке

Окно, позволяющее управлять начальными данными модели и коэффициентами ПИД-регулятора, представлено на рисунке 3.15.

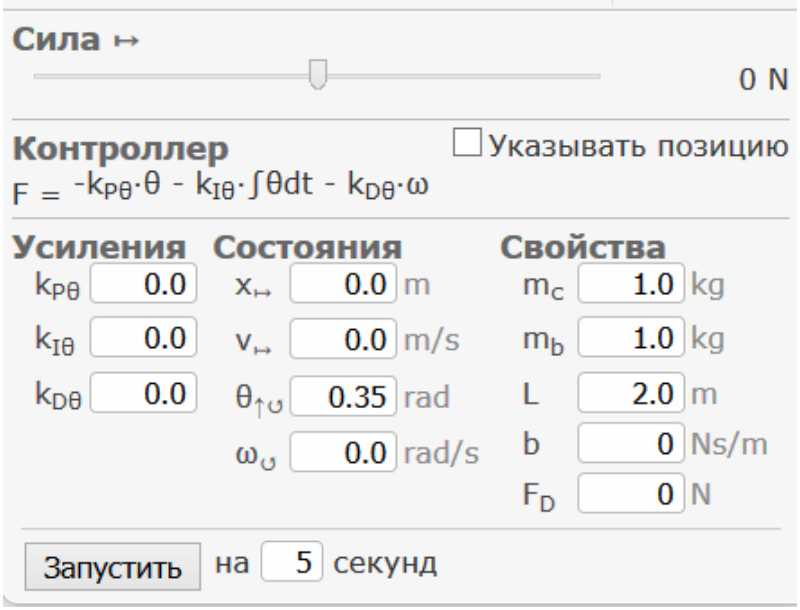

Рис. 3.15 – ПИД-регулятора

На рисунке 3.16 отображается график зависимости угла, угловой скорости и силы, воздействующей на тележку и изменения координат положения тележки от прошедшего времени.

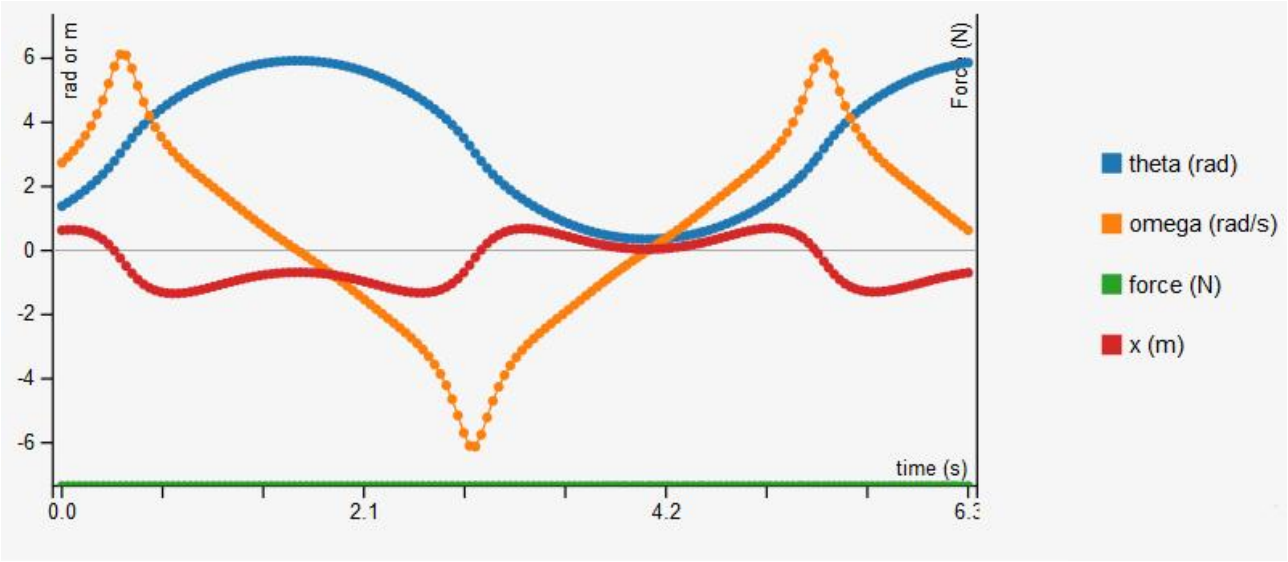

Рис. 3.16 – График изменения параметров со временем

Таким образом, была спроектирована, оптимизирована и реализованная инфраструктура виртуальной лаборатории, позволяющая независимо от технических средств разворачивать сервисы виртуальной лаборатории в минимизированной среде, расширять и использовать их. С использованием обобщенной технологии загрузки моделей были созданы примеры виртуальных моделей: движение частицы в обкладках конденсатора и обратный маятник на подвижной платформе.

3.5 Тестирование удобства использования

Поскольку основная направленность разработки виртуальной лаборатории была в сторону удобства её ИТ-инфраструктуры для дальнейшего расширения лаборатории программистами и комфортной работы пользователей, то среди всех видов тестирования остановимся на тестировании удобства использования. Как правило, во время проверки участники будут пытаться выполнять типичные задачи, наблюдатели наблюдают, слушают и делают заметки. Целью является выявление любых проблем с удобством использования, сбор качественных и количественных данных и определение удовлетворенности участников продуктом [50].

Преимущества тестирования удобства использования заключаются в том, что данная проверка позволяет командам проектирования и разработки выявлять проблемы до их кодирования. Предыдущие проблемы будут определены и исправлены, менее затратные исправления будут касаться как рабочего времени сотрудников, так и возможного влияния на график. Во время проверки удобства использования можно:

- узнать могут ли участники успешно выполнить указанные задачи;
- определить, сколько времени требуется для выполнения указанных задач;
- узнать, насколько довольны участники проверяемым веб-сайтом или другим продуктом;
- определить изменения, необходимые для повышения производительности и удовлетворенности пользователей;
- проанализировать производительность, чтобы увидеть, соответствует ли она поставленным целям.

Основные положения тестирования заключаются в том, что тестирование будет проходить от администратора виртуальной лаборатории и от лица программиста, устанавливающего окружения на рабочем компьютере. Необходимо проверить работу API, отвечающего за создание новых записей в справочнике системы, как и загружающего физические модели в формате JSON. Данные функции с точки зрения разработчика проверяется с помощью Postman collection – набора тестов, выполняющихся по сценарию программой Postman. Postman позволяет создавать коллекции интеграционных тестов, чтобы убедиться, что API работает должным образом. Тесты выполняются в определенном порядке, каждый тест запускается после завершения последнего. Для каждого теста выполняется HTTP-запрос, а утверждения, написанные на javascript, затем используются для проверки целостности кода. Результаты тестирования представлены в таблице 3.1.

Поскольку тесты и условия написаны на JavaScript, есть свобода манипулировать полученными данными по-разному, например, создавать

локальные переменные или даже создавать циклы для повторного запуска теста.

Таблица 3.1 – Основные результаты тестирования удобства использования для разработчика

Действие	Приемлемое время, секунды	Полученное время, секунды	Статус теста
Сборка VM box со всем программным обеспечением	180-600	420	Пройден
Обновление образа Docker перезагрузке образа приложения	120-300	60	Пройден
Удаление VM box	10-20	12	Пройден
Создание записи по API	2	1	Пройден
Создание записи по API	2	1	Пройден
Удаление записи по API	2	1	Пройден
Редактирование записи по API	2	2	Пройден

У Postman есть интерфейс командной строки, называемый Newman. Newman упрощает запуск набора тестов прямо из командной строки. Это легко позволяет запускать тесты Postman в системах, которые не имеют графического интерфейса, в нашем случае в контейнере Docker, но также дает возможность запускать набор тестов, написанных в Postman прямо из большинства инструментов сборки. Docker позволяет вам выполнять команды внутри самого сценария сборки, причем сценарий либо проходит, либо терпит неудачу в зависимости от результатов теста. Таким образом, во время очередной сборки проекта выполняются с помощью Newman необходимые тесты, проверяющие работоспособность проекта.

Тестирование работы пользовательского интерфейса проведем с помощью Selenium на браузере Firefox 53.0.2 (64-bit). Самым большим изменением в Selenium в последнее время стало включение WebDriver API. Принудительное управление браузером как пользователем, так и на удаленном компьютере с помощью сервера Selenium Server является большим шагом вперед с точки зрения автоматизации браузера.

Selenium WebDriver может расширять существующие языки программирования, так и к самостоятельно реализовать отдельные сценарии управления браузером. Обычно данный выпуск Selenium называют просто «WebDriver» или иногда как Selenium 2.

WebDriver разработан в более простом и более сжатом программном интерфейсе наряду с устранением некоторых ограничений в Selenium-RC API. WebDriver – это компактный объектно-ориентированный API по сравнению с Selenium1.0. Он намного эффективнее управляет браузером и преодолевает ограничения Selenium 1.x, которые влияют на функциональное тестовое покрытие, например, загрузку файлов, всплывающие окна и диалоговые окна [50].

В зависимости от нескольких факторов, включая комбинацию операционная система/браузер, WebDriver может ждать, а может ждать загрузки страницы. В некоторых случаях WebDriver может вернуть управление до того, как страница закончится или даже загрузится. Чтобы обеспечить надежность, нужно дождаться, пока элемент(ы) не будут на странице, используя явные и неявные ожидания.

Тест-кейсы на Selenium WebDriver представляют собой сценарий последовательных команд.

Тестовый пример: авторизованные пользователи могут войти на сайт.

Этапы тестирования:

1) Перейдите на сайт «http: // localhost:8080/»;

2) Введите «student1» в поле имени пользователя;

3) В поле «Пароль» введите «Testing1»;

4) Нажмите кнопку «Войти»;

5) Убедитесь, что отображается ссылка «Настройки» пользователя.

Тогда сценарий тестирования будет выглядеть как на рисунке 3.17.

```
var webdriver = require('selenium-webdriver'),
    By = require('selenium-webdriver').By,
    until = require('selenium-webdriver').until;

var driver = new webdriver.Builder()
    .forBrowser('firefox')
    .build();

driver.get(http: // localhost:8080/);
driver.findElement(By.id('login_login_username')).sendKeys('s
tudent1');
driver.findElement(By.id('login_login_password')).sendKeys('T
esting1');
driver.findElement(By.id('login_submit')).click();

driver.findElement(By.linkText('Settings')).then(function(ele
ment) {
  console.log('Yes, found the element');
}, function(error) {
  console.log('The element was not found, as expected');
});
driver.quit();
```

Рисунок 3.17 – Сценарий проверки входа на сайт

Поиск элементов пользовательского интерфейса в WebDriver может выполняться как в самом экземпляре WebDriver, так и в WebElement. Каждое из языковых привязок предоставляет элемент «Find Element» И «Find Elements» метод. Первый возвращает объект WebElement, соответствующий запросу, и генерирует исключение, если такой элемент не может быть найден. Последний возвращает список WebElements, возможно пустой, если ни один элемент DOM не соответствует запросу.

Также Selenium WebDriver можно использовать в Java-классах. На рисунке ниже представлен подобный пример кода для тестирования на приме получения заголовка (рис. 3.18).

Для начала вам нужно импортировать следующие два пакета:

- org.openqa.selenium. * - содержит класс WebDriver, необходимый для создания нового браузера, загруженного определенным драйвером;
- org.openqa.selenium.firefox.FirefoxDriver - содержит класс FirefoxDriver, необходимый для создания экземпляра конкретного для Firefox драйвера в браузере, созданном с помощью класса WebDriver.

Если ваш тест требует более сложных действий, таких как доступ к другому классу, снятие скриншотов с браузера или манипулирование внешними файлами, вам определенно нужно будет импортировать больше пакетов.

Через создание экземпляров объектов и переменных, как правило, так создается экземпляр объекта-драйвера. Класс FirefoxDriver без параметров означает, что профиль по умолчанию Firefox будет запущен Java-программой. Профиль Firefox по умолчанию аналогичен запуску Firefox в безопасном режиме (без загрузки расширений).

Для удобства сохранили базовый URL-адрес и ожидаемое название в качестве переменных. Запуск сеанса браузера начинается с метода get(), WebDriver используется для запуска нового сеанса браузера и направляет его на URL-адрес, который указан как параметр. Далее получаем фактический заголовок страницы.

У класса WebDriver есть метод getTitle(), который всегда используется для получения заголовка страницы загруженной страницы. Сравниваем ожидаемые и фактические значения. Эта часть кода просто использует базовую структуру Java if-else для сравнения фактического названия с ожидаемым.

Завершение сеанса браузера с помощью метода close(), он используется для закрытия окна браузера. Завершение работы всей программы происходит отдельно.

```
package newproject;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class PG1 {


    public static void main(String[] args) {
        // объявление и инициация объектов
        WebDriver driver ;
        System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.exe");
        driver = new FirefoxDriver();
        String baseUrl = "http://localhost:8080/";
        String expectedTitle = "Виртуальная лаборатория";
        String actualTitle = "";

        // загрузить Firefox и направить на адрес
        driver.get(baseUrl);

        // получить значение заголовка
        actualTitle = driver.getTitle();

        /*
         * сравнить полученный заголовок и ожидаемый
         * результат вывести как Test Passed или Test Failed
         */
        if (actualTitle.contentEquals(expectedTitle)){
            System.out.println("Test Passed!");
        } else {
            System.out.println("Test Failed");
        }

        //закрыть Firefox
        driver.close();

        // явно покинуть программу
        System.exit(0);
    }

}
```

Рисунок 3.18 – Пример теста получения заголовка

Если используется команда завершения без предварительного закрытия всех окон браузера, вся программа на Java закончится, оставив окно браузера открытым.

Поиск элементов в WebDriver выполняется с помощью метода findElement(By.locator()). «Локаторная» часть кода такая же, как и любой локатор, ранее использовавшийся в Selenium IDE. Вообще, рекомендуется,

чтобы элементы графического интерфейса были определены с помощью IDE и после успешной идентификации экспортировали код в WebDriver.

Далее идет пример кода (рис. 3.19), который находит элемент по его идентификатору (id). В качестве базового URL используется localhost.

```
package newproject;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class PG2 {
    public static void main(String[] args) {
        System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        String baseUrl = "http://localhost:8080/";
        String tagName = "";
        driver.get(baseUrl);
        tagName = driver.findElement(By.id("email")).getTagName();
        System.out.println(tagName);
        driver.close();
        System.exit(0);
}
}
```

Рисунок 3.19 – Пример теста для получения значения по id

Здесь использовался метод getTagName() для извлечения имени элемента разметки, чей идентификатор является «email». При запуске этот код должен иметь возможность правильно идентифицировать имя элемента разметки «input» и напечатать его в окне консоли. В таблице 3.2 подытожим различные команды, позволяющие найти элементу по разным атрибутам.

Они являются обобщенными по итогу тестирования трех виртуальных моделей физических процессов.

Результаты касаются запуска виртуальной модели с корректными данными, внесения данных большего размера, чем допускает поле модели, внесения данных недопустимого формата, изменения параметров во время работы виртуальной модели и других самых распространенных ситуаций во время использования виртуальной физической модели. По итогу работы

требования тестирования были удовлетворены, что говорит о правильной работе виртуальных моделей.

Таблица 3.2 – Варианты нахождения элемента

Варианты	Описание	Пример
By.className	Находит элементы, основанные на значении «класса» атрибута	findElement(By.className("someClassName"))
By.cssSelector	Находит элементы, основанные на движке в основе CSS	findElement(By.cssSelector("input#email"))
By.id	Находит элементы по значению индентифакатора атрибута	findElement(By.id("someId"))
By.linkText	Находит ссылку по тексту, совпадающему с заданным	findElement(By.linkText("REGISTRATION"))
By.name	Находит элементы по значению «имени» атрибута	findElement(By.name("someName"))
By.partialLinkText	Находит все элементы, содержащие данный текст-ссылку	findElement(By.partialLinkText("REG"))
By.tagName	Находит элементыпо названию элементаразметки	findElement(By.tagName("div"))
By.xpath	Находит элемент по XPath	findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]/table/tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/form/table/tbody/tr[5]"))

В виртуальных моделях и пользовательском интерфейсе много аспектов для проверки удобности и корректности, но основными результатами тестирования удобства использования интерфейса являются описанные в таблице 3.3.

Таблица 3.3 – Основные результаты тестирования удобства использования виртуальных моделей для пользователя

Действие	Ожидаемый результат	Полученный результат	Статус теста
Запуск виртуальной модели с корректными данными	Модель работает согласно заданному алгоритму с ожидаемыми результатами	Виртуальная модель работает согласно программе, результаты совпадают с ожидаемыми	Пройден
Внесение данных большего размера, чем допустимо для поля модели	Данные внести невозможно, сообщение о превышении значения	Сообщение о превышении значения, данные не вносятся	Пройден
Внесение данных недопустимого формата	Данные внести невозможно, сообщение о недопустимом формате	Сообщение о недопустимом формате, данные не вносятся	Пройден
Изменение параметров во время работы виртуальной модели	Изменить данные невозможно	Данные не поддаются изменению	Пройден
Сбрасывание значений полей к начальным параметрам	Показания становятся точно такими, как при загрузке модели	Данные возвращаются к прежним значениям	Пройден

В ходе работы было замечено, что после долгого использования виртуальные модели могут не совсем корректно отображать движущиеся детали, что является особенностью работы Firefox, однако использование браузера Google Chrom 58.0.3029.110 (64-bit), свою очередь, не приводит к подобным проблемам.

Результаты тестирования показали, что установка и использование занимают приемлемое время, а виртуальные модели позволят задать только корректные параметры.

3.6 Оценка затрат на разработку

Согласно литературе, посвященной разработке облачных технологий, они могут быть быстро подготовлены и выпущены с минимальными затратами на управление [44]. Облачные вычисления основаны на совместном использовании ресурсов для достижения согласованности и эффекта масштаба, подобно полезности сети электроснабжения [50], а так же сторонники данной технологии утверждают, что облачные вычисления позволяют компаниям избежать предварительных расходов на инфраструктуру.

Эффективность (системы) в широком смысле – это комплексная характеристика системы, отражающая степень ее соответствия потребностям и интересам ее заказчиков, пользователей, других заинтересованных лиц [18].

Программное обеспечение, которое было использовано для создания облачной инфраструктуры бесплатно и свободно распространяемое, поэтому в отношении программного обеспечения затраты не стоят внимания. Виртуальные модели также не нуждаются в закупке деталей. Это удовлетворяет предположению, что виртуальная лаборатория может оставаться бесплатной.

Если брать в расчет затраты на создание реального лабораторного оборудования, позволяющего сделать обратный маятник на тележке, например, с помощью arduino – электронной проектирующей платформы, то в сумме расходы выходят на 34600 рублей. Детали можно посмотреть в таблице 3.4.

Таблица 3.4 – Закупки для создания обратного маятника на платформе arduino

Оборудование arduino	**Стоимость, рубли**
Набор для конструирования	25000
Аккумулятор	6300
Зарядное устройство	2000
Гироскоп	2300
Итого	34600

В данной таблице идет подсчет только для одной модели, но, учитывая, что в группе может быть до 25 студентов, расходы возрастают в зависимости от того, сколько одновременно человек работает с моделью.

Напротив, виртуальная модель без подобных затрат позволит охватить сотни студентов, включая тех, что имеют удаленный доступ, без вероятности быть поломанной в результате использования. Кроме того, виртуальная лаборатория позволит с минимальными затратами загружать и расширять представленные виртуальные модели ценой лишь занимаемого места на серверном пространстве.

Таким образом, результаты тестирования разработанной виртуальной физической лаборатории показали, что она соответствует удобству использования, как программисту, так и пользователю, её эффективность оправдана, потому что виртуальная лаборатория представляет собой менее затратный способ моделирования физических явлений по сравнению с реальной лабораторией.

ЗАКЛЮЧЕНИЕ

В ходе работы над магистерской диссертацией были проанализированы статьи, посвященные виртуальным лабораториям, которые показали, что использование облачных технологий для инфраструктуры виртуальной лаборатории оправдывает использование и обеспечивает возможностью расширять качество и количество сервисов, которые предоставляет виртуальная лаборатория.

Для создания виртуальных физических моделей были выведены необходимые математические формулы, а также построен математический аппарат для приближения моделей к реальности несколькими способами.

С использованием набора практик DevOps, основанным на представлении системы как совокупности сервисов, было выстроена и оптимизирована ИТ-инфраструктура виртуальной физической лаборатории в виде частного облака, а также разработан интерфейс, позволяющий обобщенно загружать виртуальные модели.

После всех этапов проектирования и разработки созданный веб-сервис виртуальной физической лаборатории был протестирован. В ходе тестирования стало ясно, что на данном этапе реализации инфраструктура и сама «Виртуальная физическая лаборатория» в виде частного облака соответствует удобствам использования как потенциальным разработчиком, создающим новые сервисы виртуальной лаборатории, так и пользователем системы, работающим с сервисами как с черным ящиком.

В результате работы над магистерской диссертацией был создан подход к разработке облачной виртуальной лаборатории и, с помощью этого подхода, была разработана виртуальная физическая лаборатория на основе облачных технологий с примерами математических моделей физических процессов.

С помощью разработанного подхода для создания многоуровневой инфраструктуры облачной виртуальной физической лаборатории, который следует набору практик разработки программного обеспечения DevOps с

использованием прикладного программного обеспечения для построения облачных сервисов, виртуальная лаборатория приобретает такие свойства как:

- расширяемость с помощью нескольких строк кода;
- автоматическую сборку и тестирование;
- обособленную среду разработки;
- идентичное окружение для развертывания;
- удобство использования.

Данный подход к разработке инфраструктуры, основанный на использовании облачных технологий, позволит ускорить, облегчить и упростить разработку другой виртуальной лаборатории или любых сервисов, ориентированных на пользователей.

Дальнейшие развитие программного продукта вытекает из применения технологии загрузки виртуальных моделей для создания визуального редактора виртуальных моделей, также расширение библиотеки виртуальных моделей и набора функциональных возможностей для пользователей.

Разработанные подход, инфраструктура и программное обеспечение обладает практической значимостью в области прикладной математики и информатики.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Волкова В. Н. Моделирование систем и процессов : учебник для академического бакалавриата / В. Н. Волкова, Г. В. Горелова, В. Н. Козлов [и др.] – М. : Издательство Юрайт, 2015. – 449 с

2. Карпушкин, С.В. Основы моделирования процессов и систем / С.В. Карпушкин – Тамбов: Изд-во ТГТУ, 2015. – 81 с.

3. Ландау, Л.Д. Теоретическая физика. Том VIII. Электродинамика сплошных сред, 4 изд. / Л.Д. Ландау, Е.М. Лившиц – ФИЗМАТЛИТ, 2005.

4. Савельев, И.В. Курс общей физики: в 3 т. / И. В. Савельев. – М.: Лань, 2016. – 2 т.

5. Самарский, А.А. Математическое моделирование: Идеи. Методы. Примеры / А. А. Самарский, А.П. Михайлов. – М.: Физматлит. – 2002. – 320 с.

6. Борисенко, О. Д. Разработка масштабируемой программной инфраструктуры для хранения и обработки данных в задачах вычислительной биологии / О. Борисенко, А. В. Лагута, Д. Турдаков, С. Д Кузнецов// Труды ИСП РАН, т. 26, № 4, 2014 г. – с. 19-24.

7. Денисенко, В.В. ПИД регуляторы: принципы построения и модификации / В.В. Денисенко // Современные технологии автоматизации. 2007. – с. 108-114.

8. Казакевич, В. С. Система виртуальной и реальной лабораторий для физического практикума / В. С. Казакевич, С. П. Котова, А. Л. Петров, Т. Н. Сапцина, П. О. Скобелев // Известия Самарского научного центра Российской академии наук. – 2000. – №1. – с. 119-123.

9. Кравцов, Г.М. Мультимедийная виртуальная лаборатория по физике в системе дистанционного обучения / Г.М.Кравцов, Е.О.Козловский // Информационные технологии в образовании. - 2014. - № 18. - с. 80-89.

10. Сизова, Ю.В. Виртуальный лабораторный практикум на основе метода имитационного моделирования для дистанционного обучения студентов. / Н.И. Лиманова, С.Н. Потемкина, Ю.В. Сизова // Поволжский

государственный университет сервиса (ФГБОУ ВО «ПВГУС»). VI международная заочная научно-техническая конференция. Сборник статей «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. РАДИОЭЛЕКТРОНИКА. ТЕЛЕКОММУНИКАЦИИ (ITRT-2016)», 2016. – с. 37- 43.

11. Сизова, Ю.В. Разработка WEB-приложения «Виртуальная физическая лаборатория» / Ю.В. Сизова, С.Н. Потемкина, Н.И. Лиманова // Актуальные вопросы математического моделирования и разработки программного обеспечения: труды Всероссийской молодежной науч.-практ. конференции. – Тольятти: ТГУ, 2015. – с. 57- 62.

12. Сизова, Ю.В. Виртуальная физическая лаборатории для системы дистанционного обучения / Ю. В. Сизова, Н.И. Лиманова, С.Н. Потемкина. – Материалы III Международной научно-практической конференции «Математическое моделирование в сфере АПК», 2016. – с. 123-127.

13. Сизова, Ю.В. Моделирование краевых эффектов для электростатического поля плоского конденсатора / Ю.В. Сизова, А. В Розанов., С. Н Потемкина., Лиманова Н. И. // Материалы V Международной научно-практической конференции «МАТЕМАТИКА В СУЧАСНОМУ ТЕХНІЧНОМУ УНІВЕРСИТЕТІ». – 2017. – с. 96-100.

14. Соловов, А. В. Виртуальные учебные лаборатории в инженерном образовании / А. В. Соловов // Индустрия образования. – М.: МГИУ. – 2002. – с 386-392.

15. Трухин, А. В. Виды виртуальных компьютерных лабораторий [Текст] / А. В. Трухин // Открытое и дистанционное образование. – 2003. – № 03., с. 12– 20.

16. Шапошникова, Т. Л. Виртуальный лабораторный практикум в структуре информационных образовательных технологий / Т. Л. Шапошникова, Э. В. Рыкова // Ученые записки университета им. П.Ф. Лесгафта. – 2014. – №12., с. 218-221.

17. Эмирбеков Н.Э Разработка алгоритмов раскачки и стабилизации обратного маятника, закрепленного на валу двигателя / Эмирбеков Н.Э,

Эмирбеков М.Э.// АВТОМАТИКА И ПРОГРАММНАЯ ИНЖЕНЕРИЯ. – 2016. – №1., с 38-43.

18. Анисифоров, А.Б., Методики оценки эффективности информационных систем и информационных технологий в бизнесе [Электронный ресурс] / А.Б. Анисифоров, Л.О., Анисифорова // СПб. – 2014. – Режим доступа: http://elib.spbstu.ru/dl/2/3876.pdf/download/3876.pdf – (Дата обращения: 15.05.2017).

19. Виртуальная лаборатория ВиртуЛаб [Электронный ресурс] // Режим доступа: http://www.virtulab.net/ – (Дата обращения: 15.05.2017).

20. «Открытая физика», версия 2.6., часть 2 [Электронный ресурс]: Интерактивный учебник. - Электрон. дан. и прогр. - Физикон: 2005. - 1 электрон. опт. диск (CD-ROM). - Систем. требования: PC; 256 Мб RAM; 200 Мб HDD; Windows 2000/XP/Vista; 2-скоростной дисковод; дисплей; зв. карта; мышь. - Загл. с контейнера.

21. Нгуен, К.Х. Учебная виртуальная лаборатория удаленного доступа исследования биомедицинских изображений, интегрированная с расширенной системой PACS[Электронный ресурс] / К. Х. Нгуен // Fan-Nauka: международн. электрон. научн. журн. – 2013. – Режим доступа: http://www.fan-nauka.narod.ru/2013.html – (Дата обращения: 15.05.2017).

22. Berglund T. Building and Testing with Gradle / T. Berglund, M. McCullough // O'reilly, 2011.

23. Chen, L. Towards Architecting for Continuous Delivery / The 12th Working IEEE/IFIP Conference on Software Architecture, 2015.

24. Fink, J. Docker: a software as a service, operating system-level virtualization framework // Code4Lib Journal. – 2014. – T. 25.

25. Fisher. P Spring Persistence with Hibernate / P. Fisher, B. Murphy // Apress, 2016.

26. Gutierrez, F. Pro Spring Boot, Apress, 2016.

27. Mell, P. The NIST Definition of Cloud Computing / P. Mell, T. Grance // National Institute of Standards and Technology: U.S. Department of Commerce, 2011.

28. Miell, I. Docker in Practice / I. Miell, A. H. Sayers // Manning Publications Co., 2016.

29. Morris, K. Infrastructure as Code, O'Reilly Media, 2016.

30. Olausson, M. Continuous Delivery with Visual Studio ALM / M. Olausson, J. Ehn // Springer Science+Business, Media New York, 2015.

31. Soni, M. DevOps for Web Depelopment, Pact Publishing, 2016.

32. Tomson, C. Vagrant Virtual Development Environment Cookbook – Packt Publishing, 2015.

33. Vase, T. Integrating docker to a continuos delivery pipeline – a pragmatic approach, Jyväskylä, University of Jyväskylä, 2016, 66 p.

34. Walls, C. Spring in action, Manning Publications, 2015.

35. Aziz, E. An architecture for virtual laboratory experimentation / E. Aziz, S. Esche, C. Chassapis. – New Jersey: American Society for Engineering Education, 2006

36. Beltz, D. Comparing Physical, Virtual, and Hybrid Flipped Labs for General Education Biology / Beltz, D., Desharnais, R., Narguizian, P., & Son, J. // Online Learning. – 2016. – No. 3, p. 228 - 243.

37. Boettiger, C. Center for Stock Assessment Research: An introduction to Docker for reproducible research, with examples from the R environment, ArXiv, 2014.

38. Garrido, José M. Computational Models and Simulation [Text]/ José M. Garrido // Introduction to Elementary Computational Modeling. - Taylor & Francis Group, LLC – 2012., p. 57-64.

39. Haitham, A. Implementation of Cloud-based Virtual Labs for Educational Purposes IJCSNS International Journal of Computer Science and Network Security / H. A. Ali Haitham, A. El-Ghareeb, v.14, no. 7, 2014.

40. Kumar, V. A Study Virtual Laboratory: Objective, Comparison and Benefits / Vinod Kumar, Sushila Kumari // IJIACS. – 2016. – v. 5, No. 6, p. 71-73.

41. Petitpierre, C. Bottom Up Creation of a DSL Using Templates and JSON, SPLASH '11 Workshops, 2011, p. 47-52.

42. Padman, V. Design of A Virtual Laboratory for Information Assurance Education and Research / V. Padman, N. Memon. – NY.: United States Military Academy, 2012

43. Rehn, D. Tools for high-tech tool use: A framework and heuristics for using interactive simulations / Daniel A. Rehn, Emily B. Moore, Noah S. Podolefsky, Noah D. Finkelstein // Journal of Teaching and Learning with Technology, Vol. 2, No. 1, June, 2013, pp. 31 - 55.

44. Tatli, Z. Effect of a Virtual Chemistry Laboratory on Students' Achievement / Z. Tatli, A. Ayas. – Ankara: Educational Technology & Society, 2013.

45. A. Falvo, S. On XML vs JSON [Electronic recourse] // 2014. – Access mode: https://sam-falvo.github.io/2014/03/09/on-XML-vs-JSON – (Request date: 05.05.2017).

46. Bittman, B. Mind the Gap: Here Comes Hybrid Cloud [Electronic recourse] // Gartner Blog Network – 2013. – Access mode: http://blogs.gartner.com/thomas_bittman/2012/09/24/mind-the-gap-here-comes-hybrid-cloud/ – (Request date: 09.05.2017).

47. Kaewkasi, C. Cross-Platform Hybrid Cloud with Docker [Electronic recourse] // Medium.com – 2015. – Access mode: https://medium.com/@chanwit/cross-platform-hybrid-cloud-with-docker-ded000f792fb – (Request date: 15.05.2017).

48. Nunn, J. The Virtual Physical Laboratory [Electronic recourse] // 2016. – Access mode: http://www.npl.co.uk/upload/pdf/vplab_excerpt.pdf – (Request date: 15.05.2017).

49. PhET [Electronic recourse] // PhET: Free online physics, chemistry, biology, earth science and math simulations. – 2017. – Access mode: https://phet.colorado.edu – (Request date: 15.05.2017).

50. Richardson, A. Automating and Testing a REST API A Case Study in API testing using: Java, REST-assured, Postman, Tracks, cURL and HTTP Proxies [Electronic recourse] // 2017. – Access mode: http://compendiumdev.co.uk/page.php?title=tracksrestapibook – (Request date: 05.05.2017).

51. Wolfram Demonstrations [Electronic recourse] // Wolfram Demonstrations Project – 2017. – Access mode: http://demonstrations.wolfram.com/about.html – (Request date: 15.05.2017).

52. Xie, C. Molecular Workbench [Electronic recourse] // The Concord Consortium – 2013. – Access mode: http://mw.concord.org/modeler/moremw.html - 2013