МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Д.А. КРЮКОВ

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ПРЕДПРИЯТИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Москва – 2022

УДК 004.775+004.774
ББК 32.971.353
К85

Крюков Д.А. [Электронный ресурс]: Информационное обеспечение предприятия. Методические указания по выполнению практических работ / Крюков Д.А. – М.: МИРЭА – Российский технологический университет, 2022. – 1 электрон. опт. диск (CD-ROM).

Учебно-практическое издание, предназначенное для закрепления лекционного материала дисциплины Информационное обеспечение предприятия, формирования умений и владений и проверки знаний, полученных в рамках лекционных видов учебных занятий. Содержит практические задания и указания по их выполнению, способствующие усвоению материала. Подготовлено в соответствии с требованиями ФГОС ВО по направлению подготовки 09.03.04 «Программная инженерия». Предназначено для студентов, обучающихся по направлению подготовки бакалавров 09.03.04 «Программная инженерия» всех форм обучения, а также может быть полезно для студентов, обучающихся по другим направлениям и специальностям.

Методические указания издаются в авторской редакции.

Автор**:** Крюков Дмитрий Алексеевич

Рецензенты:
Писаренко Юрий Андрианович, д.т.н., проф. кафедры ХТООС, ИТХТ имени М.В. Ломоносова

Минимальные системные требования:
Наличие операционной системы Windows, поддерживаемой производителем.
Наличие свободного места в оперативной памяти не менее 128 Мб.
Наличие свободного места в памяти хранения (на жестком диске) не менее 30 Мб.
Наличие интерфейса ввода информации.
Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).
Подписано к использованию по решению Редакционно-издательского совета МИРЭА – Российского технологического университета от ___ _____ 2022 г.
Объем ___ Мб
Тираж 10

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Информационное обеспечение предприятия» предназначена для формирования у обучающихся необходимых знаний о месте и роли глобальных информационных ресурсов в цифровую эпоху, направлена на получение знаний об основных теоретических понятиях, технологических особенностях и подходах к разработке компонентов информационного обеспечения современных предприятий, их интеграции в информационно-телекоммуникационную сеть «Интернет». Дисциплина «Информационное обеспечение предприятия» является обязательной дисциплиной вариативной части блока «Дисциплины» учебного плана направления подготовки бакалавров 09.03.04 «Программная инженерия». Целью данной дисциплины является формирование у обучающихся профессиональных ОПК-1, ОПК-4, ПК-15 компетенции в соответствии с требованиями ФГОС ВО по направлению подготовки 09.03.04 «Программная инженерия». В результате изучения дисциплины обучающийся должен:

знать:

- основные концепции, принципы, теории и факты, связанные с информатикой;
- принципы осуществления поиска, хранения, обработки и анализа информации из различных источников и баз данных, представления ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий;
- принципы подготовки презентаций, оформления научно-технических отчетов по результатам выполненной работы, опубликования результатов исследований в виде статей и докладов на научно-технических конференциях.

уметь:

- применять основные концепции, принципы, теории и факты, связанные с информатикой,
- осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий;
- готовить презентации, оформлять научно-технические отчеты по результатам выполненной работы, публиковать результаты исследований в виде статей и докладов на научно-технических конференциях.

владеть:

- основными концепциями, принципами, теориями и фактами, связанными с информатикой;

- поиском, хранением, обработкой и анализом информации из различных источников и баз данных, представлением ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий.
- навыками подготовки презентации, оформления научно-технических отчетов по результатам выполненной работы, опубликования результатов исследований в виде статей и докладов на научно-технических конференциях.

Настоящий практикум предназначен для обеспечения проверки полученных теоретических знаний, формирования умений и владений компетенций посредством текущего контроля.

Целью текущего контроля знаний является установление подробной, реальной картины студенческих достижений и успешности усвоения ими бакалаврской учебной программы на данный момент времени.

1. ОБЩИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Практические работы, выполняемые студентами 4 курса по дисциплине «Информационное обеспечение предприятия», занимают важное место в учебном процессе освоения дисциплин по кафедре «Корпоративные информационные системы». Практические работы выполняется студентами самостоятельно. Результаты, полученные в ходе выполнения работы, имеют учебный характер, но, вместе с тем, должны раскрывать теоретические и практические аспекты квалификации будущего бакалавра.

1.1. Условия выполнения практических работ

Практические работы выполняются студентами в компьютерных классах либо на персональных компьютерах. По результатам выполнения всех практических заданий готовится общий либо индивидуальный отчеты, в котором учитываются результаты выполнения всех задач и полученные при этом навыки. При выполнении практических работ рекомендуется следовать плану-графику, представленному в настоящем методическом пособии.

Для выполнения практических заданий необходимо наличие следующего программного обеспечения:

- любая среда разработки: Visual Studio с поддержкой C#, ASP.net, Visual Studio Code, IDE MonoDevelop (для Linux и MacOS), Eclipse;
- вспомогательные инструменты web-разработчика: JetBrains PyCharm Community (для Python), IntelliJ IDEA Community, Java Development Kit,

Java Runtime Environment (для Java), библиотеки BeautifulSoup4, requests, JSOUP;

– любая сборка программного обеспечения для построения клиент-серверной инфраструктуры: XAMPP, Денвер и т.п.;
– прикладное ПО: Docker, cURL, Kubernetes;
– современный web-браузер;
– ER-проектирование: MySQL Workbench.

1.2. Цели выполнения практических работ

Практические работы предназначены для получения навыков умения и владения теоретическими знаниями, полученными студентами в процессе изучения дисциплины «Информационное обеспечение предприятия», а также других дисциплин кафедры «Корпоративные информационные системы»: «Технологии программирования», «Управление данными», «Информационные сети», «Открытые информационные системы», «Проектирование информационных систем».

Целью практических работ является приобретение студентом навыков самостоятельного развертывания макета инфраструктуры клиент-серверных решений, разработки сценариев работы с базой данных, технологий асинхронной обработки запросов, работы с интерфейсами прикладного программирования, современными подходами к взаимодействию с информационными ресурсами и провайдерами данных, использования штатных диагностических средств web-разработчика.

1.3. Задачи, подлежащие решению в ходе выполнения практических работ

Для полноценного освоения учебной программы студенту необходимо выполнить все практические задания настоящего методического пособия. Выполнение заданий предполагает последовательное решение следующих задач:

– усвоение и анализ теоретического лекционного материала;
– освоение языка гипертекстовой разметки и каскадных таблиц стилей;
– выбор языка разработки сценариев на стороне сервера, освоение его в объеме, требуемом заданиями;
– последовательное решение задач практических заданий;
– подготовка обобщенного либо индивидуального отчета о выполнении практических заданий согласно установленным требованиям.

1.4. План-график выполнения практических работ

При выполнении практических работ рекомендуется придерживаться плана-графика, представленного в таблице. Следует учитывать, что для полноценного освоения учебной программы необходимо распределять трудоемкость на аудиторную (контактную) и на самостоятельную работу. При этом доля самостоятельной работы при выполнении каждого задания должна быть выше доли аудиторной работы в два раза.

Согласно приведенному плану-графику на выполнение шестнадцати практических работ отводится полный семестр. Практические работы №№1-6 являются менее трудоемкими чем, практические работы №№7-17. Форма отчетности выбирается преподавателем: индивидуальные отчеты по каждой практической работе либо общий отчет о выполнении практикума. Подготовку общего отчета целесообразно начинать с 9 недели семестра. Сдача отчета будет организована в течение последних трех недель семестра. Отступать от плана-графика не рекомендуется, при этом возможно досрочное выполнение плана-графика.

№ п/п	Неделя семестра	Содержание этапа
1.	1 неделя	Выполнение практической работы №1 Выполнение практической работы №2
2.	2 неделя	Выполнение практической работы №3 Выполнение практической работы №4
3.	3 неделя	Выполнение практической работы №5 Выполнение практической работы №6
4.	4 неделя	Выполнение практической работы №7 Выполнение практической работы №8
5.	5 неделя	Выполнение практической работы №9 Выполнение практической работы №10
6.	6-7 неделя	Выполнение практической работы №11
7.	8 неделя	Выполнение практической работы №12
8.	9 неделя	Выполнение практической работы №13 Подготовка отчета
9.	10-11 неделя	Выполнение практической работы №14 Подготовка отчета
10.	12 неделя	Выполнение практической работы №15 Подготовка отчета
11.	13 неделя	Выполнение практической работы №16 Подготовка отчета
12.	14-15 недели	Выполнение практической работы №17 Подготовка отчета
13.	16 неделя	Сдача отчета

2. СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ РАБОТ

Далее изложено содержание заданий, требования к отчету и методические указания по выполнению практических работ. Задания упорядочены в соответствии с принципом от простого к сложному. Порядок заданий синхронизирован с лекционным материалом.

2.1. Практическая работа №1

Освоить язык расширенных запросов выбранной поисковой системы, овладеть возможностями семантического поиска.

Общие сведения

Информационный поиск – процесс поиска неструктурированной документальной информации, удовлетворяющей информационные потребности.

Представляет собой процесс выявления в некотором множестве документов (текстов), которые соответствуют указанной теме (предмету), удовлетворяют (релевантны) заранее определенному условию поиска (запросу).

Разновидности информационного поиска:

- адресный поиск (поиск URL),
- семантический поиск (поиск ответов на вопросы),
- документальный поиск (поиск URN),
- фактографический поиск (вертикальный поиск),
- поиск по содержимому (компьютерное зрение),
- метапоиск (мультипоиск по нескольким источникам).

Семантический поиск - процесс поиска документов по их содержанию и значению. Принципиальная разница между адресным и семантическим поисками состоит в том, что при адресном поиске документ рассматривается как объект с точки зрения формы, а при семантическом поиске — с точки зрения содержания.

При семантическом поиске находится множество документов без указания адресов, которые формализуются в виде развернутого ответа на вопрос либо в единый сводный документ.

Расширенный поиск – совокупность операторов, уточняющих текстовые критерии поиска информации, обогащающих текстовое описание поиска с помощью языка поисковых запросов.

Как правило, язык поисковых запросов реализует следующие возможности уточнения искомых поисковых образов документов:

- исключение слов,
- связка посредством логического «И», «ИЛИ»,

- точное совпадение,
- установка расстояния между словами.

А также ограничения области поиска:

- конкретным адресом,
- типом документов,
- интервалом времени,
- языком.

Пример расширенного запроса в поисковую систему «Яндекс» приведен ниже.

```
часы & +наручные & +мужские
&& ("Frederique Constant" | !Longines | "Auguste Reymond")
&& (цена | стоимость | прайс | рубли | магазин)
~~(подделка && копия && реплика && неоригинал)
+(механические | механика)
```

Задачи

1. Определить опытную поисковую систему.
2. Ознакомиться с документацией к языку расширенных запросов выбранной поисковой системы.
3. Привести несколько примеров расширенных запросов, содержащих операторы «И», «ИЛИ», «НЕ», скобки, из области интересов студента. Изложить их практический смысл.
4. Ознакомиться с критериями, которым должны удовлетворять системы семантического поиска.
5. Определиться с двумя опытными семантическими поисковыми системами.
6. Провести сравнение двух выбранных систем используя следующие метрики: *достоверность* и *полнота*. Критерий *достоверности* исчисляется бинарной логикой: если ответ семантической системы является правильным и не содержит ошибок, то принимает значение 1, иначе 0. *Полнота* – отражает количество информации, несущей некий смысл сообщения, или объем содержащихся в нем знаний по отношению к общему объему сообщения. Нечеткий субъективный критерий, определяется студентом по шкале от 0% до 100%. Относительность предполагает распределение долей полноты ответов двух поисковых систем, в сумме образующих не более 100% для одного запроса. Ответ системы может называется содержательным, а оценка превышать 0% в том случае, если помимо ответа на задачу семантического поиска, будет представлена дополнительная полезная информация относительно предмета поиска. В случае, если обе семантических поисковых систем возвращают идентично содержательный ответ, то 100% смысловой

содержательности разделяется между ними пополам. В случае, если обе системы не ответили на вопрос, то суммарное значение их смысловой содержательности может составлять 0%.

7. Выполнить не менее 5-ти запросов в поисковые системы для оценки средней достоверности и полноты. В запросах могут содержаться математические формулы, сравнения (оператор vs.), прямые вопросы, требующие логического вывода. Следует учитывать, что большинство систем семантического поиска ориентированы на английский язык. Результат представить в форме таблицы:

Запросы	Семантическая система 1		Семантическая система 2	
	Достоверность	Полнота*	Достоверность	Полнота*
Запрос 1	1	70%	1	30%
Запрос 2	1	50%	1	50%
Запрос 3	0	0%	1	100%
Запрос 4	1	70%	0	30%
Запрос 5	0	10%	1	90%
	Среднее	Среднее	Среднее	Среднее

*суммарная полнота одного запроса не может превышать 100%

8. Подготовить материалы в отчет о выполнении практического курса занятий.

2.2. Практическая работа №2

Освоить работу с платформой Docker, приобрести навыки самостоятельной разработки, развёртывания и запуска приложений в контейнерах.

Общие сведения

Одним из основных понятий технологии Docker является «контейнер». Контейнер - это стандартная единица программного обеспечения, которая упаковывает код и все его зависимости, поэтому приложение работает быстро и надежно из одной вычислительной среды в другую. Образ контейнера Docker - это легкий, автономный исполняемый пакет программного обеспечения, который включает в себя все необходимое для запуска приложения: код, среду выполнения, системные инструменты, системные библиотеки и настройки.

Docker – совокупность программных средств для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет объединить приложение с окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему, а также обеспечить среду управления контейнерами.

В состав программных средств входит демон - сервер контейнеров, клиентские средства, позволяющие из интерфейса командной строки управлять образами и контейнерами, а также API, позволяющий в парадигме REST управлять контейнерами.

В отличие от виртуализации, Docker обеспечивает одновременную работу нескольких операционных систем на одном компьютере, в то время, как Docker-контейнеры используют одно и то же ядро операционной системы и изолируют процессы приложения от остальной системы.

Организация бизнес-процессов средствами виртуальных машин сопряжена со следующими недостатками:

– неэффективное использование ресурсов, так как каждая виртуальная машина представляет собой полноценную операционную систему;
– зависимость от платформы;
– медленное и требовательное к ресурсам масштабирование решения, основанного на виртуальных машинах.

Если для решения аналогичной задачи применить контейнеры, то, в сравнении с виртуальными машинами, можно будет получить следующие преимущества:

– эффективное использование ресурсов;
– независимость от платформы;
– легковесное развёртывание за счёт использования слоёв образов.

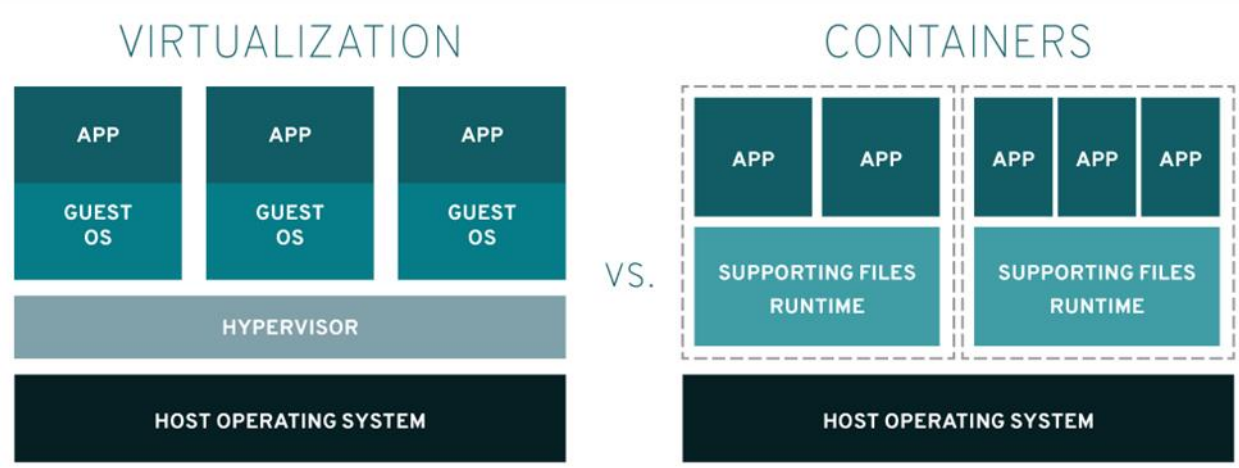

Рис. 1. Сравнение структуры виртуализации и контейнеризации

К преимуществам Docker относятся:

– ускоренный процесс разработки. Нет необходимости устанавливать вспомогательные инструменты вроде PostgreSQL, Redis, Elasticsearch: их можно запускать в контейнерах;
– инкапсуляция приложений;
– мониторинг;
– масштабирование.

Контейнерное программное обеспечение, доступное как для Linux, так и для Windows, всегда будет работать одинаково, независимо от инфраструктуры. Контейнеры изолируют программное обеспечение от его среды и обеспечивают его работу.

Важным понятием технологии Docker являются слои. Это составляющие любого Docker-контейнера, например, операционная система, исполняемые файлы, скрипты, прикладное программное обеспечение. Фактически, набор слоев образуют Docker-контейнер. Каждый слой описывает изменение состояния образа в сравнении с тем состоянием, в котором он пребывал до добавления следующего слоя.

Задачи

1. Создать учетную запись на ресурсе разработчиков DockerHub.
2. Определить материнскую (host) операционную систему (Windows[1], Linux, Mac OS) для развёртывания контейнеров Docker.
3. Ознакомиться с документацией по развертыванию ПО Docker на выбранной операционной системе.
4. Проверить статуса демона/службы Docker.
5. Проверить список существующих контейнеров.
6. Определить образ операционной системы для контейнера.
7. Получить образ выбранной операционной системы и версии с репозитория DockerHub.
8. Запустить образ, создать контейнер Docker из одного слоя – операционной системы.
9. Создать контейнер из двух слоев. Первый слой – базовый, операционная система. Второй слой скрипт выполнения команды «Hello world», например, /bin/echo 'Hello world'.
10. Добавить в контейнер прикладное программное обеспечение, например, образ web-сервера nginx.
11. Изучить инструкции DockerFile.
12. Создать с использованием инструкций DockerFile новый контейнер, содержащий не менее пяти слоев.
13. Проверить список существующих контейнеров.
14. Подготовить материалы в отчет о выполнении практического курса занятий.

[1] Для установки Docker в ОС Windows обязательно наличие компонента «Microsoft Hyper-V». Проверка наличия и активности компонента осуществляется командной systeminfo.

2.3. Практическая работа №3

Освоить работу со штатными диагностическими инструментами современных браузеров (т.н. консоль разработчика), позволяющими осуществлять профилирование HTTP-запросов, проверку дерева содержания, дерева отображения и иные возможности.

Общие сведения

Web-клиент (далее – браузер) – общедоступное программное обеспечение, предназначенное для взаимодействия с web-серверами, DNS-, FTP-серверами, обеспечивающее получение и обработку информационных и служебных данных и формирование web-страниц в соответствии со спецификациями W3C.

Основное предназначение браузера – отображать веб-ресурсы. Для этого браузер формирует совокупность HTTP-запросов к ресурсам, расположенным на web-сервере, объединяет ответы, формирует web-страницу и отрисовывает её. Одним из компонентов браузера является модуль отображения. Модуль отвечает за синтаксический анализ кода HTML и CSS, построение дерева отображения и дерева содержания.

В Firefox применяется модуль отображения Gecko (Quantum) – собственная разработка Mozilla, в Safari и Chrome используется WebKit. Chrome использует несколько экземпляров модуля отображения, по одному для каждой вкладки.

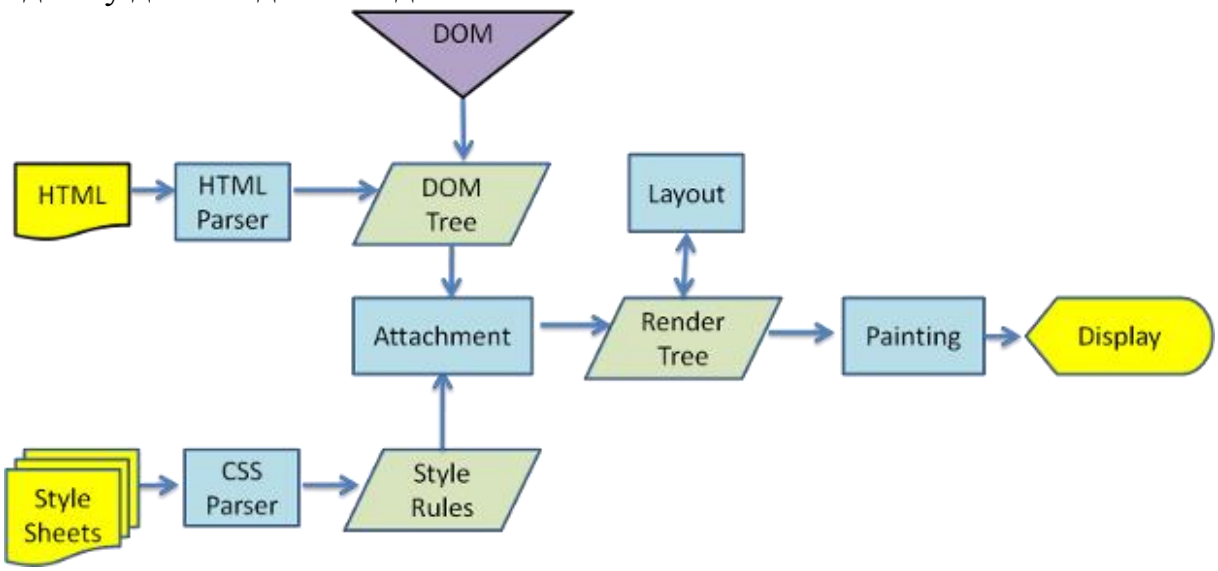

Рис. 2. Блок-схема работы модуля отображения

Модуль отображения:

– получает содержание запрошенного документа по протоколу сетевого уровня;

– выполняет синтаксический анализ HTML-документа;

– переводит теги в узлы DOM;
– строит дерево содержания;
– выполняет анализ CSS-файлов;
– строит дерево отображения.

Консоль разработчика в браузере – это инструмент, благодаря которому web-разработчик может просмотреть наполнение страницы, выводимой браузером, обнаружить и отладить ошибки, возникающие в ходе функционирования веб-страницы. Например, консоль в браузере Firefox позволяет осуществлять поиск по активному содержимому, отображать метки времени возникновения сообщений, позволяет фильтровать сообщения по типу и месту возникновения, исполнять и просматривать вывод JavaScript-кода.

Задачи

1. Используя один из современных браузеров (Firefox, Chrome, Яндекс и т.п.) изучить функционал штатного диагностического инструмента – панели (консоли) для разработчиков. Спектр доступных инструментов в разных браузерах может отличаться. Чаще всего встречаются следующие инструменты: просмотр элементов web-страницы, профилирование сетевой активности, просмотр cookies, консоль для отладки JavaScript и редактор CSS-стилей;
2. Определить перечень доступных инструментов панели разработчиков в выбранном браузере;
3. Установить назначение и провести анализ возможностей каждого инструмента, определить степень полезности в web-разработке;
4. Подготовить материалы в отчет о выполнении практического курса занятий.

2.4. Практическая работа №4

Изучить принципы построения серверной инфраструктуры для развертывания web-приложений. Получить практические навыки по установке и настройке программного обеспечения локального сервера на примере сборки XAMPP или иной по выбору студента.

Общие сведения

XAMPP аббревиатура кроссплатформенной сборки программного обеспечения сервера, включающей web-сервер Apache, СУБД MySQL, интерпретатор скриптов PHP, Perl и набор дополнительных библиотек, позволяющих развернуть полноценный web-сервер.

XAMPP работает со всеми ОС Microsoft, Linux, Mac OS, Solaris. Программа свободно распространяется и является бесплатным, удобным в работе web-сервером, способным обслуживать динамические страницы. XAMPP является одной из популярных сборок, с помощью которой возможно быстро развернуть полноценный и быстрый web-сервер.

Apache HTTP Server - web-сервер, который обеспечивает работу около 46% сайтов по всему миру. Работа web-сервера заключается в обслуживании посетителей web-сайтов. Для этого он выполняет роль посредника между сервером и компьютерами клиентов, обеспечивает асинхронную обработку каждого запроса пользователя. Дополнительно web-сервер при помощи соответствующих интерпретаторов исполняет скрипты, написанные на различных языках программирования, таких как PHP, Python, Perl и других. В результате работы скриптов может быть сформирован статический HTML-файл для передачи в браузер пользователя.

Например, посетитель обращается на страницу web-сайта, при этом браузер направляет запрос на web-сервер и Apache возвращает ответ со всеми запрошенными ресурсами: текст, стили, изображения и иные. Сервер и клиент взаимодействуют по протоколу HTTP и Apache ответственен за гладкое и безопасное соединение между двумя машинами.

MySQL - это сервер управления баз данных, используемый в разных приложениях. Из преимуществ СУБД MySQL стоит отметить простоту использования, гибкость, низкую стоимость владения (относительно платных СУБД), а также масштабируемость и производительность.

PHP - это распространенный язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработок и его код может внедряться непосредственно в HTML. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов. Популярность в области построения web-сайтов определяется наличием большого набора встроенных языковых средств и дополнительных модулей для разработки web-приложений. Основные из них:

– автоматическое извлечение POST и GET-параметров, а также переменных окружения web-сервера в предопределённые массивы;
– взаимодействие с большим количеством различных систем управления базами данных через дополнительные модули;
– автоматизированная отправка HTTP-заголовков;
– работа с HTTP-авторизацией;
– работа с cookies и сессиями;
– работа с локальными и удалёнными файлами, сокетами;
– обработка файлов, загружаемых на сервер;
– работа с XForms.

phpMyAdmin – специальное web-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД MySQL. phpMyAdmin позволяет через браузер и не только осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Приложение пользуется популярностью у web-разработчиков, так как позволяет управлять СУБД MySQL без непосредственного использования SQL-конструкций.

Задачи

1. Получить инсталлятор XAMPP с официального сайта;
2. Установить сборку по пути «C:/xampp» компонентами: Apache, MySQL, PHP, phpMyAdmin;
3. После установки запустить компоненты Apache и MySQL;
4. Проверить работу web-сервера Apache: открыть браузер и проверить адрес «http://localhost/». Получить уведомление об успешной установке и запуске;
5. Определить корневую директорию тестового web-сайта. В корневой директории создать простой тестовый сайт и начальный файл web-сайта index.php, содержащий вывод сообщения;
6. Проверить работоспособность web-сайта, перейдя в браузере по его адресу;
7. С использованием phpMyAdmin создать новую базу данных, таблицу с двумя столбцами. Первый столбец: тип данных - integer, индекс – primary, auto_increment. Второй столбец типа text;
8. Наполнить таблицу записями;
9. Добавить в код файла index.php скрипт, содержащий инструкции подключения к СУБД, выполнения запроса select …, сохранения результата запроса в массив, вывода записей на web-страницу, отключения от СУБД.
10. Подготовить материалы в отчет о выполнении практического курса занятий.

2.5. Практическая работа №5

Получить навыки реализации взаимодействия программного кода с базой данных в web-приложениях. Освоить инструмент визуального проектирования баз данных MySQL Workbench.

Общие сведения

Для удобства взаимодействия с базами данных и их содержимым на сервере баз данных MySQL целесообразно использовать инструмент для визуального проектирования баз данных MySQL Workbench. Программа может быть инсталлирована в виде компонента MySQL. Также существует вариант автономной установки.

Возможности Workbench:

- наглядно представить модель базы данных в графическом виде.
- реализовать функциональный механизм установки связей между таблицами, в том числе «многие ко многим» с созданием таблицы связей.
- восстанавливать структуры таблиц из уже существующей на сервере баз данных.
- имеет редактор SQL-запросов, позволяющий направлять их серверу и получать ответ в виде таблицы.

Задачи

1. На базе инфраструктуры, полученной в результате выполнения практического задания №4, определить скриптовый язык для реализации взаимодействия с базой данных: PHP, Perl, Pyton, NodeJS, ASP.net и др. Определить СУБД: MySQL, SQLite, PostgreSQL, MariaDB, Firebird;
2. Добавить в интерпретатор скриптового языка соответствующую библиотеку для работы с СУБД, импортировать библиотеку в код;
3. Реализовать в скрипте функции подключения к СУБД, закрытия соединения, Создания базы данных, создания таблиц базы данных, наполнение таблиц баз данных, вывод с объединением данных в таблицах;
4. Создать две таблицы, объединенные связью «многие ко многим». Например, автомобили-водители, отели-туристы, заказчики-исполнители, фильмы-актеры, писатели-книги и т.п. Создать таблицу-справочник для связи «один ко многим»;
5. Наполнить таблицы баз данных осмысленными значениями;
6. Выполнить с помощью скрипта примеры вывода данных, объединенных по трем таблицам;
7. Составить схему базы данных с использованием MySQL Workbench;
8. Подготовить материалы в отчет о выполнении практического курса занятий.

2.6. Практическая работа №6

Изучить возможности диагностической программы cURL. Получить практические навыки по взаимодействию с серверами с использованием командной строки.

Общие сведения

cURL – кроссплатформенная служебная программа, функционирующая в режиме командной строки, позволяющая взаимодействовать с различными серверами по множеству различных протоколов. Программа cURL может автоматизировать передачу файлов или последовательность таких операций. Является эффективным программным средством моделирования действий пользователя в web-браузере и диагностики программного обеспечения сервера.

Программа поддерживает протоколы: FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, LDAP, а также POP3, IMAP и SMTP. Также cURL поддерживает методы HTTP POST, HTTP PUT, загрузку на FTP, загрузку через формы HTTP.

Команда cURL может быть выполнена с различными ключами. Ниже приведены те ключи, которые потребуются для выполнения задания. Регистр ключей имеет значение.

Таблица 1.

Ключ	Описание
-A	Задает строку с названием агента для отправки на HTTP сервер. Для кодирования пробелов в строке в начале и конце строки необходимо установить одинарные кавычки. Данное значение может быть установлено параметром **-H, --header.** *Если данный параметр в командной строке встречается несколько раз, то рабочим будет последний.*
-b	Передавать данные на HTTP сервер как куки (cookie), как будто это данные ранее полученные от сервера в строке "Set-Cookie:". Данные должны быть в формате "NAME1=VALUE1; NAME2=VALUE2". Если в строке не используются символы '=', то она рассматривается в качестве имени файла, где хранится строка из coockie, и которая в случае совпадения должна быть использована. При использовании данного метода также активируется "cookie parser", который сделает запись входящих cookie. Это может быть удобно в сочетании с параметром -L, --location. Формат файлов с куками должен быть либо в формате Netscape/Mozilla, либо хранить простые заголовки HTTP. *Замечание*: файл, определенный параметром -b, --cookie используется только для чтения, для записи куков используйте параметр -c, --cookie-jar, а для записи HTTP заголовков используется -D, --dump-header! *Если данный параметр описан несколько раз, то использоваться*

	будет последний.
-c	Задать файл, в который будут записаны все куки после завершения команды. Туда будут записаны все прочитанные куки из файла и полученные от удаленного сервера. Если куки не будут получены и не считывались параметров --cookie <name=data>, то файл не будет записан. Формат записываемого файла Netscape. Если вместо имени файла задать тире '-', то куки будут выводиться в стандартный вывод. В случае если куки невозможно создать или записать curl не сообщит об этом явным образом. Необходимо использовать параметр **-v**, который выведет предупреждение, но это будет единственная информация о проблеме.
-d	Отправляет указанный в data POST запрос на сервер HTTP аналогично тому, как пользователь, заполнив HTML форму, нажал кнопку "отправить". Curl передаст данные на сервер с помощью *content-type application/x-www-form-urlen-coded.* Параметр --data <data> аналогичен **--data-ascii**, для отправки двоичных данных необходимо использовать параметр **--data-binary**. Для url перекодировки значений полей формы можно использовать **--data-urlencode**. В случае если данный параметр будут встречен в командной строке несколько раз, то данные будут объединены, используя разделительный символ '&'. Например, строка *'-d name=daniel -d skill=lousy'* будет передана в следующем виде *'name=daniel&skill=lousy'*. Если значение параметра начинается с "**@**" то после него должно быть имя файла с данными или дефис '-' в случае если вы хотите, чтобы curl использовал данные из стандартного ввода. Можно указать несколько файлов, но их содержимое должно быть подготовлено правильным образом.
-e	Отправляет информацию "Referer Page" на HTTP сервер. Это можно сделать и параметром **-H, --header flag**. При использовании **-L, --location** можно добавить ";auto" в --referer URL. *Если данный параметр описан несколько раз, то использоваться будет последний.*
-G	При использовании данного параметра все данные описанные в **d, --data** или **--data-binary** будут использовать GET запрос, вместо POST. Данные будут добавлены через разделитель '?'. Если использовать вместе с параметром -i, POST данные будут добавлены в HEAD запроса.
-I	Включить вывод HTTP заголовков, они включают в себя такую информацию как : имя сервера, дату документа, версию http и многое другое.
-I	Выбирать только HTTP заголовки. У HTTP серверов есть команда HEAD, которая используется только для получения заголовков. При использовании FTP и FILE, curl покажет только размер файла и время последнего его изменения.
-H	Использовать дополнительные заголовки при получении веб-страницы, можно определить в curl любое количество дополнительных заголовков. Замечание: Если добавить пользовательский заголовок, имеющий

	одинаковое имя с внутренним, уже имеющимся в curl заголовком, то будет использоваться пользовательский. Поэтому, необходимо отдавать себе отчет в действиях, когда вы подменяете заголовки. Удалить внутренний заголовок, заменив его собственным не меняя содержимого, можно указав его, справа от двоеточия, как в **-H "Host:"**. Если вы отправляете свой заголовок без содержимого, он должен заканчиваться точкой с запятой, то есть *-H "X-Custom-Header;"* отправит *"X-Custom-Header:"*. Curl будет проверять, что каждый отправленный заголовок имеет правильный маркер конца строки. Вы не должны добавлять контент, как часть заголовка: вы не должны добавлять символы новой строки или возврата каретки, они создадут проблемы. Как вариант можно использовать **-A, --user-agent** и **-e, --referer**. *Данный параметр может быть использован несколько раз, чтобы добавить, заменить или удалить несколько заголовков.*
-L	Если сервер сообщает, что запрашиваемая страница была перемещена на другой адрес (указывается заголовком и кодом ответе 3XX), данный параметр заставляет curl сделать запрос по новому адресу. Если используются параметры **-i, --include** или-**I, --head**, будут показаны заголовки всех запрошенных страниц. Если используется авторизация, данные посылаются только для начального хоста и если переадресация происходит на другой хост, то он не сможет перехватить данные авторизации. Для изменения передачи данных авторизации используется параметр **--location-trusted**, для задания количества переходов **--max-redirs**.

-o	Записывает вывод в файл (file) вместо стандартного вывода. Если в URL адресе используются перечисления или диапазоны, то можно использовать символ '#' с числом. Эта переменная будет заменена, как например:
	`curl http://{one,two}.site.com -o "file_#1.txt"`
	или использовать несколько переменных, как:
	`curl http://{site,host}.host[1-5].com -o "#1_#2"`
	Этот параметр можно использовать столько раз, сколько есть URL адресов в команде.
	См. также **--create-dirs** для создания папок.

-x	Использовать указанный HTTP прокси. Если номер порта не задан, то использовать порт 1080. Этот параметр отменяет существующие переменные среды, установленные для использования прокси сервера. Все операции, которые выполняются над HTTP-прокси будут прозрачно преобразованы в HTTP. Это означает, что определенные протоколы и определенные операции могут быть недоступны. Например, нельзя использовать туннель для прокси-сервера, для этого существует **-p, --proxytunnel.** *Если данный параметр указан несколько раз, то использоваться будет последний.*

Задачи

1. Получите исполняемый файл cURL и установите его в операционной системе;
2. Отправить GET-запрос на любой web-сайт;
3. Сохранить содержимое web-сайта в файл;
4. Выбрать прокси сервер. Например, 5.164.128.64:53281
5. Отправить GET-запрос на любой web-сайт через выбранный прокси сервер;
6. Сохранить в файл содержимое web-сайта, полученное через выбранный прокси сервер с эмуляцией вариантов User Agent (десктопный web-браузер, мобильный web-браузер, поисковый робот). Сравнить содержимое файлов с ответами, сделать выводы.
7. Отправить GET-запрос на адрес «https:/facebook.com» с помощью браузера, а затем с помощью cURL. Каковы результаты? Просмотреть HTTP-заголовки с помощью cURL и установить причину по которой cURL возвращает пустой ответ. Проверить заголовки запроса повторно, выполнив GET-запрос с перенаправлением.
8. С помощью консоли разработчика браузера установить наименование полей формы ввода информации об авторизации любого web-сайта. Выполнить GET-запрос с авторизацией на любом сайте с помощью cURL. Выполнить POST-запрос с авторизацией на любом сайте с помощью cURL. При необходимости используйте параметры работы с Cookie;
9. Подготовить материалы в отчет о выполнении практического курса занятий.

2.7. Практическая работа №7

Овладеть навыками использования HTTP-запросов GET, POST и технологии AJAX при проектировании web-сайта для перемещения между страницами сайта.

Общие сведения

HTTP – HyperText Transfer Protocol, протокол передачи гипертекста, сетевой протокол прикладного уровня, изначально предназначенный для передачи гипертекстовых документов. Благодаря протоколу HTTP обеспечивается работа сети Интернет. Основой HTTP является парадигма «клиент-сервер».

Метод HTTP – последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию на web-сервере. Основные методы HTTP: GET, HEAD, POST, PUT.

Метод GET используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс, например, поиска. В этом случае в тело ответного сообщения следует включить информацию о ходе и о результатах выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:

```
GET /search?q=http+is+stateless HTTP/1.1
```

Метод POST применяется для передачи пользовательских данных заданному ресурсу. В полях ввода информации web-страницы пользователи обычно могут оставлять запись (комментарий, запрос), после чего они передаются серверу методом POST. Передаваемые данные включаются в тело запроса. С помощью POST можно так же загружать файлы на web-сервер.

AJAX (Asynchronous JavaScript and XML) асинхронный JavaScript и XML - подход к построению интерактивных пользовательских интерфейсов веб-приложений, позволяющих выполнять асинхронные динамические запросы к серверу без видимой перезагрузки веб-страницы

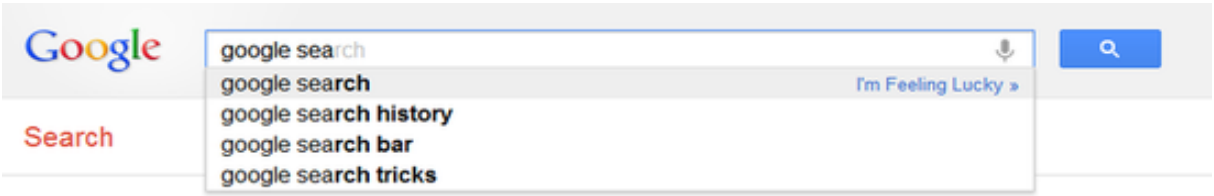

Рис. 3. Результат применения AJAX

Преимуществами использования AJAX являются:

– сокращение трафика при работе с web-сайтом,
– уменьшение нагрузки на web-сервер в разы,
– ускорение реакции web-страницы без обновления,
– интерактивная обработка данных на web-сайте.

В качестве формата контейнеров передачи данных могут использоваться фрагменты простого текста, HTML-кода, JSON или XML.

Задание

1. С использованием инфраструктуры, полученной в результате выполнения практического задания №4, создать web-страницу index1, содержащую форму, состоящую из двух полей ввода и кнопки. По нажатии на кнопку осуществляется переход на вторую web-страницу index2. При кодировании страниц

допускается использовать любой язык программирования скриптов на стороне сервера;
2. На web-странице index2 реализовать вывод содержимого первого и второго поля формы ввода index1;
3. Реализовать передачу вводимых значений от страницы index1 к странице index2 двумя способами: с помощью метода POST и метода GET;
4. Создать web-страницу index3, содержащую однострочное текстовое полей ввода и многострочное поле вывода;
5. Создать web-страницу index4, которая будет осуществлять обработку AJAX-запросов от index3, Обработка заключается в регистронезависимом поиске в массиве слов по совпадению их начальных букв;
6. Массив необходимо заранее наполнить значениями. Допускается использовать таблицу базы данных вместо массива;
7. В поле ввода web-страницы index3 необходимо вводить начальные буквы элементов массива. Поле вывода обновлять полными именами записей массива (базы данных), совпадающим по начальным буквам;
8. Взаимодействие между index3 и index4 организовать при помощи технологии AJAX без перезагрузки страницы в соответствии со схемой на рис. 4;
9. Подготовить материалы в отчет о выполнении практического курса занятий.

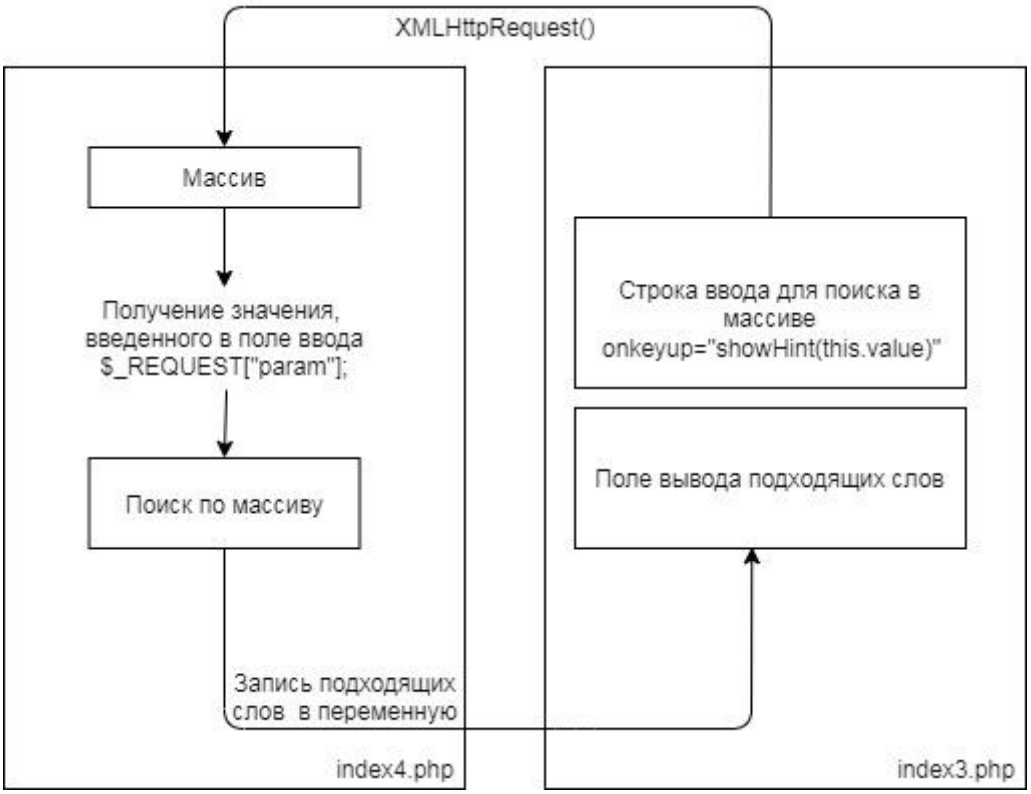

Рис. 4. Схема взаимодействия скриптов с применением технологии AJAX

2.8. Практическая работа №8

Получить навыки работы с API информационных ресурсов с применением парадигмы взаимодействия REST.

Общие сведения

Интерфейс прикладного программирования (application programming interface, API) - набор готовых классов, процедур, функций, структур и констант, предоставляемых информационным ресурсом для использования во внешних программных продуктах. Используется для интеграции сторонних приложений. API определяет допустимую функциональность, которую предоставляет информационный ресурс для внешних взаимодействий. API позволяет абстрагироваться от того, как именно эта функциональность реализована.

REST (Representational State Transfer) передача репрезентативного состояния — концепция взаимодействия компонентов клиент-серверного приложения. В парадигме REST количество методов и сложность протокола строго ограничены, из-за чего количество отдельных сервисов (URI) может быть большим. Все RESTful приложения основаны на технологии манипуляции объектами CRUD (Create, Read, Update, Delete), что в протоколе HTTP будет выражено методами PUT, GET, POST, DELETE. В REST-сервисах акцент сделан на доступ к ресурсам, а не на исполнение удаленных методов.

Задачи

1. Определить информационный ресурс для взаимодействия посредством API. Например: Вконтакте, Яндекс, Википедия, Wolfram Alpha, NASA, Open Food Facts, Merriam-Webster Dictionary, WeatherBit, Numbers и другие. Пройти регистрацию, получить AppID разработчика. Данный идентификатор потребуется включать во все HTTP-запросы;
2. Ознакомиться с документацией на API информационного ресурса. Убедиться в возможности организации взаимодействия с использованием подхода REST;
3. Выполнить посредством cURL, web-браузера, скрипта на web-сайте несколько примеров из документации. Убедиться в отсутствии ошибок в ответных сообщениях;
4. Изучить формат ответных сообщений. Локализовать переменную с результатом;

5. С использованием инфраструктуры, полученной в результате выполнения практического задания №4, создать web-страницу, содержащую поле ввода запроса, поле вывода ответа, элемент управления. По нажатии на элемент управления осуществлять отправку запроса в информационный ресурс с использованием API. Предусмотреть обработку ответа и вывод ответа в поле вывода, в качестве критерия запроса возможно использовать арифметические операции. При создании GET-запросов необходимо учитывать корректность кодирования символов URL-адресов;
6. Подготовить материалы в отчет о выполнении практического курса занятий.

2.9. Практическая работа №9

Получить навыки работы с API текстовых информационных ресурсов-классификаторов. Освоить возможности языков сериализации данных.

Общие сведения

Информационный ресурс — упорядоченная совокупность данных, представляющих индивидуальные и коллективные экспертные знания, отдельные документы, отдельные массивы документов, составляющие базы и банки данных, библиотеки, архивы, фонды, информационные системы и другие системы определенной предметной тематической области.

Информационный ресурс классификатор - систематизированный перечень именованных объектов, каждому из которых в соответствие дан уникальный код. Применяется в автоматизированных системах управления и в системах цифровой обработки информации.

Является стандартным кодовым языком документов, финансовых отчётов и автоматизированных систем. Примерами ресурсов-классификаторов являются:

- UDC, Universal Decimal Classification (в России локализован как УДК, Универсальный десятичный классификатор);
- ISO 3166 alpha 2, alpha 3;
- Классификатор адресов Российской Федерации (КЛАДР);
- Общероссийский классификатор административно-территориального деления объектов (ОКАТО);
- Society for Worldwide Interbank Financial Telecommunications, SWIFT (в России локализован как БИК, банковский идентификационный код).

Сериализация - процесс перевода какой-либо структуры данных, объекта в специальную форму, поддерживающую обратный процесс десериализации. Например, для сохранения и передачи по каналу связи. Примерами языков сериализации данных являются: XML, JSON, ASN.1, Google Boost, Google Proto Buffers и т.д. Рассмотрим XML и JSON.

XML, eXtensible Markup Language — расширяемый язык разметки. XML разрабатывался как язык с простым формальным синтаксисом, удобным для создания и обработки документов программами. Язык XML не определяет разметку, используемую в документах: разработчик может создать теги в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

JSON, JavaScript Object Notation — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Используется для сериализации сложных структур. Является более лаконичным чем XML. JSON-текст представляет собой совокупность следующих структур:

- Набора пар ключ:значение (объект, запись, структура, словарь, хэш-таблица, список с ключом). Ключом может быть только строка, значением — любая форма.
- Упорядоченного набора значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Задание

1. С использованием инфраструктуры, полученной в результате выполнения практического задания №4, создать web-страницу, содержащую поисковую строку для ввода критерия поиска финансовой организации по БИК и SWIFT (рис. 5.a), поля вывода информации о финансовой организации, элемент управления. Предусмотреть структурирование ответа (рис. 5.b);
2. Определить провайдера, предоставляющего доступ через API, позволяющий реализовать функцию поиска финансовой организации. Пройти регистрацию, получить API-ключ разработчика. Данный идентификатор потребуется включать во все HTTP-запросы;
3. По нажатии на элемент управления осуществляется отправка HTTP-запроса методом POST в информационный ресурс с использованием API, обработка JSON/XML-ответа и вывод полученных сведений в поле отчета;
4. Подготовить материалы в отчет о выполнении практического курса занятий.

Рис. 5. Форма ввода и вывода сведений по БИК

2.10. Практическая работа №10

Получить навыки работы с API объектографических информационных ресурсов на примере взаимодействия с провайдером ФИАС. Закрепить навыки использования технологии AJAX и языков сериализации данных.

Общие сведения

Информационный ресурс — упорядоченная совокупность данных, представляющих индивидуальные и коллективные экспертные знания, отдельные документы, отдельные массивы документов, составляющие базы и банки данных, библиотеки, архивы, фонды, информационные системы и другие системы определенной предметной тематической области.

В фактографических информационных ресурсах находятся записи о состоянии внешнего мира, информация об интересующих объектах предметной области в виде фактов. Следующие категории данных являются фактографическими: биографические данные о сотрудниках, сведения о выпуске продукции, справочная информация определенной отрасли, бизнес-аналитика, инфографика и т.п. Фактографические ресурсы можно разделить на две категории: объектографические и статистические.

Объектографический информационный ресурс - фактографический ресурс, записи в котором содержат данные об отдельном объекте предметной области. Например:

- БД нарушений и нарушителей ПДД;
- Федеральная информационная адресная система (ФИАС);

- Единый государственный реестр прав на недвижимое имущество и сделок с ним;
- Справочники местоположений Wi-Fi точек доступа, базовых станций GSM, IP-адресов;
- Единый государственный реестр юридических лиц;
- Единый государственный реестр налогоплательщиков;
- Ресурсы регистраторов доменных имен.

Статистический информационный ресурс - фактографический ресурс, содержащий данные о свойствах событий, процессов и явлений (содержит данные о той или иной количественной характеристике объекта). Например:

- Базы данных рейтинговых агентств (Alexa, Stadard&Poor’s);
- База данных Федеральной службы государственной статистики (Росстат);
- Котировки фондовых бирж (NASDAQ, nyse.com);
- Базы данных ключевых показателей эффективности (Key Performance Indicators, KPI).

Задачи

1. С использованием инфраструктуры, полученной в результате выполнения практического задания №4, создать web-страницу, содержащую поисковую строку для ввода критерия поиска географического адреса (рис. 6.a), поля вывода информации об индексе, кодах ФИАС, КЛАДР, элемент управления. Предусмотреть структурирование ответа (рис. 6.b);
2. Реализовать выпадающий список вариантов автоматического дополнения (подсказок) географического адреса без обновления страницы или фрейма с использованием технологии AJAX (рис. 6.c);
3. Определить провайдера, предоставляющего доступ через API, позволяющего реализовать функцию поиска адреса в Федеральной информационной адресной системе. Пройти регистрацию, получить API-ключ разработчика. Данный идентификатор потребуется включать во все HTTP-запросы;
4. При вводе поискового запроса должна осуществляться отправка HTTP-запросов методом POST в информационный ресурс с использованием API, обработка JSON/XML-ответа и вывод полученных сведений в список автоматического дополнения и в поле отчета по окончании ввода;
5. Подготовить материалы в отчет о выполнении практического курса занятий.

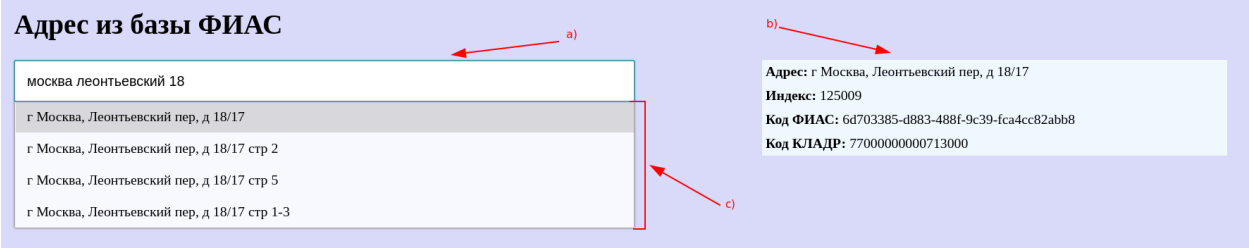

Рис. 6. Форма ввода и вывода информации из ФИАС

2.11. Практическая работа №11*

Получить навыки работы с RSS. Освоить возможности инструментов автоматического парсинга данных из RSS-лент новостных сайтов и записи содержимого в базу данных.

Общие сведения

RSS (Rich Site Summary, обогащённая сводка сайта) - семейство XML-форматов, предназначенных для описания лент новостей, анонсов статей, изменений в блогах и т. п. Начиная с версии 2.0 аббревиатура поменяла расшифровку на Really Simple Syndication – очень простое структурирование.

Задачи

1. Изучить структуру RSS-лент новостных сайтов. Изучить функции библиотек парсинга BeautifulSoup (Python) или JSoup (Java);
2. Определить новостного провайдера, имеющего RSS-ленту. Например, https://1prime.ru/export/rss2/index.xml. Определить тег записи для новости (для http://1prime.ru – <item>);
3. Определить поля, которыми задаются свойства новости (класса-элемента RSS) (рис. 7), Опишите поля: наименование, тип;
4. Составить ER-диаграмму класса-элемента RSS;
5. Определить необходимые библиотеки для парсинга RSS выбранным языком программирования;
6. Загрузить RSS-описание с использованием HTTP-request (для библиотек requests и bs4);
7. Структурировать данные RSS-ленты с применением библиотеки для парсинга html-страниц/xml-документов;
8. Проверить правильность выполнения п.7 путем вывода первой новости из ленты RSS;
9. Написать парсер RSS-лент, использующий список экземпляров класса (объектов) для хранения считанной информации;

10. Написать класс для хранения элементов ленты с конструктором, использующим данные ER-диаграммы. В классе предусмотреть метод вывода всех полей класса в строку: toString(self);
11. Реализовать вывод парсера в список экземпляров (объектов класса)
12. Написать метод загрузки данных в БД. Заполнить базу данных новостей;
13. Подготовить материалы в отчет о выполнении практического курса занятий.

Рис. 7. Свойства класса-элемента RSS-ленты

2.12. Практическая работа №12*

Получить навыки работы с промежуточными форматами хранения структур данных на примере форматов JSON, XML, CSV и XLS, в целях представления результатов парсинга данных.

Общие сведения

JSON (JavaScript Object Notation) — текстовый формат обмена данными. JSON-текст представляет собой (в закодированном виде) одну из двух структур:

- Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка, значением — любая форма. Повторяющиеся имена ключей допустимы, но не рекомендуются стандартом; обработка таких ситуаций происходит на усмотрение программного обеспечения;
- Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

За счёт своей лаконичности по сравнению с XML формат JSON может быть более подходящим для хранения (сериализации) сложных структур. Применяется в веб-приложениях как для обмена данными между

браузером и сервером (AJAX), так и между серверами (программные HTTP-сопряжения)

XML (англ. eXtensible Markup Language) — расширяемый язык разметки. Используется для хранения и передачи данных, например, в SOAP-запросах. XML представляет собой сочетание двух структур:

- тегов, заключенных в угловые скобки (теги могут начинаться либо с буквы, либо с символа подчеркивания);
- атрибутов (правила задания имен для атрибутов те же, что и для тегов).

CSV – текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми.

Хотя разделителем значений колонок является символ запятой, на практике часто используются другие разделители.

Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("). Если в значении встречаются кавычки, то они представляются в файле в виде двух кавычек подряд.

XLS это текстовый формат хранения документов Microsoft Excel в теговой модели (с 2007 года формат хранения по умолчанию - XLSX). В отличие от вышеописанных форматов, работа с XLS по умолчанию не реализовывается в большинстве языков программирования, поэтому для работы с ним используются сторонние библиотеки, например, SimpleXLSX, phpspreadsheet, PHPExcel.

Существует множество иных форматов хранения структур данных в том числе, поддерживающих сериализацию: Protocol buffers, ASN.1, YAML, Thrift, MessagePack и другие, в том числе, нативные алгоритмы, встроенные в языки программирования.

Задачи

1. С использованием результатов, полученных по итогам выполнения практического задания №12, создать web-страницу, содержащую возможность выбора диапазона дат и выбора целевого формата сохранения результатов парсинга. Определить, как минимум три формата на выбор студента.
2. Реализовать сохранение данных в каждом формате. Задействовать при необходимости библиотеки для работы с перечисленными форматами хранения данных.

3. По нажатии на элемент управления осуществлять отправку HTTP-запроса на получение данных нужного формата в рамках заданного временного интервала.
4. Продемонстрировать результаты сохранения данных в браузере или в ином подходящем приложении.
5. Подготовить материалы в отчет о выполнении практического курса занятий.

2.13. Практическая работа №13

Получить навыки решения первичных вопросов обеспечения информационной безопасности создаваемых продуктов, поддерживающих многопользовательский режим представления полномочий и данных.

Общие сведения

С доступом к информации и ресурсам системы связана группа таких важных понятий, как идентификация, аутентификация, авторизация. Порядок обеспечения авторизации представляется следующей последовательностью шагов:

1. Присвоение идентификатора. Каждый субъект или пользователь системы (сети, ресурса) имеет некоторую уникальную информацию (число, наименование, строку символов), идентифицирующую субъект, Эта информация является идентификатором субъекта системы (сети, ресурса). Пользователь, имеющий зарегистрированный идентификатор, является законным (легальным) субъектом.
2. Сравнение идентификатора. Идентификация - это присвоение субъектам и объектам доступа идентификатора и сравнение предъявляемого пользователем идентификатора с перечнем присвоенных идентификаторов. Таким образом, идентификация субъекта это процедура распознавания субъекта по его идентификатору.
3. Подтверждение подлинности. Следующим шагом взаимодействия системы с субъектом является аутентификация субъекта. Аутентификация субъекта - это проверка принадлежности субъекту доступа предъявленного им идентификатора. Процедура аутентификации устанавливает, является ли субъект подлинным, именно тем, кем он себя объявил.
4. Предоставление доступа. После идентификации и аутентификации субъекта выполняется процедура авторизации. Авторизацией субъекта является процедура предоставления законному субъекту,

успешно прошедшему идентификацию и авторизацию соответствующих полномочий и доступных ресурсов системы.

Каждый из перечисленных шагов нуждается в применении процедур обеспечения информационной безопасности. В рамках данной работы предлагается освоить процедуры хэширования парольной информации и противодействия внедрению SQL-кода.

Хэширование - это метод криптозащиты, представляющий собой контрольное преобразование информации из данных неограниченного размера. Путем выполнения криптографических преобразований вычисляется хэш-значение фиксированной длины, однозначно соответствующее исходным данным.

Такое криптографическое контрольное суммирование широко используется в различных методах защиты информации, в частности, для подтверждения целостности данных. Кроме того, данный метод применяется в схемах электронной подписи: «подписывается» обычно хэш-значение данных, а не все данные целиком, а также в схемах аутентификации пользователей.

Хэширование используется крайне часто. Самые примитивные хэш функции – деление с остатком по модулю, выбор первой буквы в алфавите в любом телефоном справочнике. В схемах аутентификации пользователей. В базе данных сохраняется хэш-значение пароля/идентификатора вместо его первоначального вида. Поскольку хэш-функция – необратимое преобразование, то во время восстановления пароля его сбрасывают, а во время взлома подменяют хэш. Для решения криптографических задач применимы криптографически стойкие хэш-функции имеющие следующие особенности:

однонаправленность - стойкость к вычислению прообраза. Устойчивость к коллизии 1-го рода - вычислительно невозможно подобрать сообщение, имеющее хэш-значение равное другому сообщению, устойчивость к коллизии 2-го рода - вычислительно невозможно подобрать сообщения, имеющих одинаковое хэш-значение.

avalanche (лавинный) effect - незначительное изменение аргумента функции должно существенно менять хэш-значение. Функция удовлетворяет лавинному критерию, если изменение одного бита на входе вызывает изменение половины выходных битов.

Примерами криптографических функций являются:

- ГОСТ Р 34.11-2012 («Стрибог») - отечественный алгоритм выработки хэш-функции. Размер хэша 256 или 512 бит;
- MD6 (Message Digest), размер хэша является параметром, но не может быть больше 512 бит;
- Whirlpool (от одного из разработчиков AES), размер хэша до 512 бит;
- SHA-3 (Keccak), размер хэша: 224, 256, 384, 512 бит.

Несмотря на стойкость криптографических функций к вычислению прообраза либо подбор аргументов, имеющих идентичные хэш-значения, актуальна проблема применения радужных таблиц для атак на криптографические функции. С целью противодействия и усложнения проведения данной атаки необходимо искусственно удлинять парольную строку с помощью модификатора, называемого солью. Это строка случайных символов, которые подается на вход хэш-функции вместе с исходными данными. Осложняет восстановление группы исходных паролей за один проход полного перебора или с помощью предварительно построенных радужных таблиц. Соль, как правило, не является секретной, и хранится рядом с хэшем пароля в открытом виде (т.н. статическая соль). Секретным является то, как именно эта соль применятся. Динамическая соль - подстрока или определенные символы исходного (случайного) пароля или хэша от пароля, например, на языке php:

```
Hash = (md5(md5($DynSalt.$password).$StatSalt)
```

Внедрение SQL-кода (SQL-инъекция) - один из распространённых способов атаки приложения, работающие с базами данных, основанный на добавлении в запрос произвольного SQL-кода. Атака позволяет выполнить действие, которое не планировалось создателем скрипта. Инъекция появляется из входящих данных, которые не фильтруются алгоритмами проверки вводимых данных. Существует несколько способов защиты от данной атаки. В задании предлагается определить меры защиты самостоятельно, ознакомившись предварительно с предлагаемыми для конкретной СУБД либо языка программирования.

Задачи

6. Определить язык программирования и СУБД, для реализации задач.
7. Определить целевую систему, в которой будет создан функционал авторизации. В качестве основы возможно использование результатов, полученных в других заданиях практикума.
8. Реализовать в СУБД необходимые структуры для авторизации пользователей.
9. Реализовать форму регистрации/аутентификации учетных записей в системе, «привязать» форму к СУБД.
10. Реализовать функционал регистрации учетных записей. Учесть необходимость проверки существования пользователя с введённым набором данных.
11. Реализовать функционал аутентификации и авторизации пользователей, с использованием обращений к СУБД. Учесть

требования информационной безопасности по хэшированию чувствительных данных с солью.

12. При помощи конструкций выбранного языка либо инструментов СУБД «защитить» SQL-запросы любым из доступных методов предотвращения внедрения SQL-кода.
13. Подготовить материалы в отчет о выполнении практического курса занятий.

2.14. Практическая работа №14*

Изучить возможности web-сервера nginx в рамках кэширования запросов. Получить практические навыки по установке и настройке конфигурации web-сервера на примере nginx.

Общие сведения

Nginx — это HTTP-сервер и обратный прокси-сервер, почтовый прокси-сервер, а также TCP/UDP прокси-сервер общего назначения.

В отличие от других продуктов данного сегмента, Nginx использует иной принцип обработки входящих данных. Он разбивает каждый запрос пользователя на несколько атомарных, упрощая таким образом, обработку каждого. В терминологии Nginx они получили название рабочее соединение.

После обработки каждое соединение собирается в одном виртуальном контейнере, чтобы трансформироваться в единый первоначальный запрос, а после отправляется пользователю. Одно соединение может одновременно обрабатывать до 1024 запросов конечного пользователя.

Для уменьшения нагрузки на оперативную память веб-сервер использует выделенный сегмент памяти, который называется «пул». Он динамический и расширяется при увеличении длины запроса.

Основная функциональность HTTP-сервера:

- Обслуживание статических запросов, индексных файлов, автоматическое создание списка файлов, кэш дескрипторов открытых файлов;
- Акселерированное обратное проксирование с кэшированием, распределение нагрузки и отказоустойчивостью;
- Поддержка FastCGI, uwsgi, SCGI и memcached серверов с кэшированием, распределение нагрузки и отказоустойчивость;
- Модульность, фильтры, в том числе сжатие gzip, byte-ranges (докачка), chunked-ответы, XSLT-фильтр, SSI-фильтр, преобразование изображений; несколько подзапросов на одной

странице, обрабатываемые в SSI-фильтре через прокси или FastCGI/uwsgi/SCGI, выполняются параллельно;
- Поддержка TLS и расширения eSNI;
- Поддержка HTTP/2 с приоритезацией на основе весов и зависимостей.

Кэш – это высокоскоростной уровень хранения, на котором требуемый набор данных, как правило, временного характера. Доступ к данным на этом уровне осуществляется значительно быстрее, чем к основному месту их хранения. С помощью кэширования становится возможным повторное использование ранее полученных или вычисленных данных.

Кэширование ответов позволяет уменьшить количество запросов к проксируемому серверу, снижая его нагрузку и увеличивая скорость получения ответа. В случае кэширования ответов сторонних серверов (например, информационных ресурсов) это поможет избежать превышения ограничений на количество запросов.

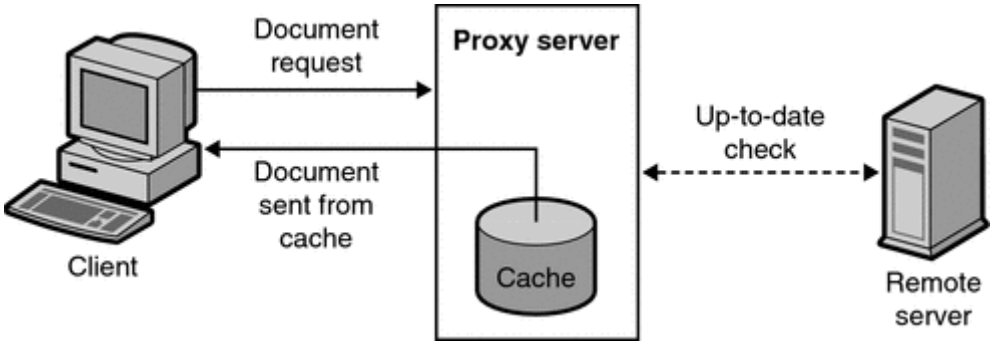

Рис. 8. Схема работы прокси-сервера с возможностью кэширования ответов

Задачи

1. Определить операционную систему для установки web-сервера nginx. В качестве основы возможно использование результатов, полученных в других заданиях практикума;
2. Ознакомиться с документацией по установке nginx на выбранной операционной системе;
3. Скачать и установить nginx. Проверить статус службы nginx. Проверить работу web-сервера nginx посредством браузера;
4. Используя результаты практической работы №8, изменить конфигурацию web-сервера, превратив его в прокси-сервер. В качестве проксируемого сервера задействовать информационный ресурс, предоставляющий API;
5. Проверить работоспособность прокси-сервера;

6. Внести изменения в код, заменив URL информационного ресурса на адрес прокси-сервера;
7. Изменить конфигурацию, включив кэширование ответов, получаемых с информационного ресурса. Определить время хранения кэша и размер хранилища кэша, исключить кэширование данных с использованием cookie.
8. Убедиться в работоспособности кэширования;
9. Проверить работу прокси-сервера с использованием измененного web-приложения;
10. Подготовить материалы в отчет о выполнении практического курса занятий.

2.15. Практическая работа №15

Получить навыки работы с REST API при помощи вспомогательного инструмента – Postman.

Общие сведения

Representational State Transfer (REST) в переводе — это передача состояния представления. Технология позволяет получать и модифицировать данные и состояние удаленных приложений, передавая их в форме HTTP-команд.

Application Programming Interface (API), программный интерфейс приложения — это набор инструментов, который позволяет одним программам работать с другими. API предусматривает, что программы могут работать в том числе и на разных компьютерах. В этом случае требуется организовать интерфейс API так, чтобы ПО могло запрашивать функции друг друга удаленно.

REST API — это прикладной программный интерфейс (API), который использует HTTP-запросы для получения, извлечения, размещения и удаления данных. Интерфейс представлен четырьмя классическими методами:

- GET — метод чтения информации. GET-запросы всегда только возвращают данные с сервера, и никогда их не меняют и не удаляют.
- POST — создание новых записей.
- PUT — редактирование записей.
- DELETE — удаление записей.

Архитектура REST API достаточно популярное решение для организации взаимодействия между различными программами. Причины состоят в том, что HTTP-протокол реализован во всех языках программирования и всех операционных системах, в отличие от проприетарных протоколов.

Чаще всего REST API применяют для:

- связи мобильных приложений с серверными.
- построения микросервисных серверных приложений. Это архитектурный подход, при котором большие приложения разбиваются на много маленьких частей.
- предоставления доступа к программам сторонних разработчиков. Например, Stripe API позволяет программистам встраивать обработку платежей в свои приложения.

Тестирование API — это разновидность тестирования программного обеспечения, которая определяет, соответствует ли недавно разработанное приложение ожиданиям с точки зрения надежности, производительности, операций и безопасности.

При проведении тестирования имеют значение категории кодов ответов HTTP-сервера:

- 1xx (100 — 199): ответ имеет информационный характер;
- 2xx (200 — 299): успешный ответ;
- 3xx (300 — 399): следует принять дальнейшие меры для выполнения запроса;
- 4xx (400 — 499): запрос не может быть завершен;
- 5xx (500 — 599): сервер не имеет возможности выполнить запрос.

Postman — это набор инструментов для тестирования API. Он является средой разработки для создания, тестирования, контроля и публикации API-документации. Назначение Postman — тестирование отправки запросов с клиента на сервер и получения ответа от сервера. Фактически это HTTP-клиент для тестирования веб-сайтов. С помощью него можно составлять и редактировать простые или сложные HTTP-запросы. Составленные запросы автоматически сохраняются на будущее для повторного применения. Ответы сервера так же могут сохраняться.

Программа снабжена встроенным редактором запросов, с возможностями кодирования запросов, загрузки из файла и отправки бинарных данных. История запросов сохраняется по категориям (коллекциям). При коллаборативной работе имеется возможность открыть совместный доступ к какой-нибудь из коллекций.

Задачи

1. Скачать клиентское приложение с официального web-сайта https://www.getpostman.com/.
2. Изучить пользовательскую документацию приложения, интерфейс программы: вкладки с запросом; метод, который будет использован для запроса; URL-адрес; параметры запроса/ответа, коллекции, параметры.

3. Определить целевой API для проведения тестов.
4. Отправить тестовый запрос, используя метод GET. Проверить тело ответа и статус.
5. Используя практическую работу №8, протестировать API веб-сайта используя метод POST. Составить тело и параметры запроса. Проверить тело ответа и статус.
6. Создать коллекцию добавив в нее запросы с GET и POST HTTP-методами.
7. Подготовить материалы в отчет о выполнении практического курса занятий.

2.16. Практическая работа №16

Изучить способы и протоколы аутентификации пользователя. Получить навыки реализации сервера с аутентификацией пользователя одним из возможных способов.

Общие сведения

Аутентификация — предоставление доказательств, что вы на самом деле есть тот, кем идентифицировались (от слова “authentic” — истинный, подлинный).

Существует несколько методов аутентификации:

- аутентификация по паролю;
- HTTP аутентификация;
- аутентификация по токенам.

Аутентификация по паролю основана на том, что пользователь должен предоставить наименование учетной записи и пароль для успешной идентификации и аутентификации в системе. Пара username/password задается пользователем при его регистрации в системе, при этом в качестве имени пользователя может выступать адрес электронной почты пользователя.

Применительно к веб-приложениям, существует несколько стандартных протоколов для аутентификации по паролю.

HTTP аутентификация

Этот протокол, описанный в стандартах HTTP 1.0/1.1, существует очень давно и до сих пор активно применяется в корпоративной среде. Применительно к веб-сайтам работает следующим образом:

1. Сервер, при обращении неавторизованного клиента к защищенному ресурсу, отсылает HTTP статус “401 Unauthorized” и добавляет заголовок “WWW-Authenticate” с указанием схемы и параметров аутентификации.

2. Браузер, при получении такого ответа, автоматически показывает диалог ввода имени учетной записи и пароля. Пользователь вводит детали своей учетной записи.
3. Во всех последующих запросах к этому веб-сайту браузер автоматически добавляет HTTP заголовок “Authorization”, в котором передаются данные пользователя для аутентификации сервером.
4. Сервер аутентифицирует пользователя по данным из этого заголовка. Решение о предоставлении доступа (авторизация) производится отдельно на основании роли пользователя, ACL или других данных учетной записи.

Аутентификация по токенам

Такой способ аутентификации чаще всего применяется при построении распределенных систем Single Sign-On (SSO), где одно приложение (т.н. service provider или relying party) делегирует функцию аутентификации пользователей другому приложению, обычно именуемому identity provider или authentication service. Типичный пример этого способа — вход в приложение через учетную запись в социальных сетях. Здесь социальные сети являются сервисами аутентификации, а приложение доверяет функцию аутентификации пользователей социальным сетям.

Реализация этого способа заключается в том, что identity provider (IP) предоставляет достоверные сведения о пользователе в виде токена, а service provider (SP) приложение использует этот токен для идентификации, аутентификации и авторизации пользователя.

На общем уровне, весь процесс выглядит следующим образом:

1. клиент аутентифицируется в identity provider одним из способов, специфичным для него (пароль, ключ доступа, сертификат, Kerberos, и т.д.).
2. клиент просит identity provider предоставить ему токен для конкретного SP-приложения.
3. identity provider генерирует токен и отправляет его клиенту.
4. клиент аутентифицируется в SP-приложении при помощи этого токена.

Существует несколько распространенных форматов токенов для веб-приложений:

Simple Web Token (SWT) — наиболее простой формат, представляющий собой набор произвольных пар имя/значение в формате кодирования HTML. Токен подписывается с помощью симметричного ключа, таким образом оба IP- и SP-приложения должны иметь этот ключ для возможности создания/проверки токена.

JSON Web Token (JWT) — содержит три блока, разделенных точками: заголовок, набор полей (claims) и подпись. Первые два блока представлены в JSON-формате и дополнительно закодированы в формат base64. Набор полей содержит произвольные пары имя/значения. Подпись

может генерироваться при помощи и симметричных алгоритмов шифрования, и асимметричных. Кроме того, существует отдельный стандарт, отписывающий формат зашифрованного JWT-токена.

Security Assertion Markup Language (SAML) — определяет токены (SAML assertions) в XML-формате, включающем информацию об эмитенте, о субъекте, необходимые условия для проверки токена, набор дополнительных утверждений (statements) о пользователе. Подпись SAML-токенов осуществляется при помощи ассиметричной криптографии. Кроме того, в отличие от предыдущих форматов, SAML-токены содержат механизм для подтверждения владения токеном, что позволяет предотвратить перехват токенов через man-in-the-middle-атаки при использовании незащищенных соединений.

Задачи

1. Изучить способы и протоколы аутентификации. Выбрать один из протоколов для реализации механизма аутентификации по токенам.
2. Выбрать язык программирования для создания сервера. Определить необходимые библиотеки для создания сервера и реализации механизма аутентификации.
3. Подключить базу данных для хранения данных пользователя, создать структуры базы данных (путем миграции, скрипта и т.п.)
4. Реализовать для пользователя регистрацию, авторизацию и действие, требующее аутентификации.
5. С помощью средств тестирования (Postman, SoapUI и т.п.) продемонстрировать результат регистрации и авторизацию.
6. Продемонстрировать результат выполнения аутентифицированного и не аутентифицированного запросов.
7. Подготовить материалы в отчет о выполнении практического курса занятий.

2.17. Практическая работа №17*

Освоить работу с платформой Kubernetes, приобрести навыки самостоятельной разработки, развёртывания и запуска приложений из нескольких реплик.

Общие сведения

Kubernetes разработка инженеров Google. Компания одной из первых начала работу с контейнерами и публично заявляла, что всё в Google

работает в контейнерах. Google генерирует более 2 миллиардов развертываний в неделю – все они работают на Borg и Omega - внутренних системах Google. В 2014 году Google открыл исходный код проекта Kubernetes, который вобрал в себя лучшее из названных систем. Но прошло 7 лет и K8S (он же Kubernetes) стал успешным open-source проектом, который используется по всему миру. В большинстве компаний инфраструктура построена на K8S - от маленьких проектов до огромных кластеров и облаков, которые обеспечивают бесперебойную доступность сервисов. Слово «Kubernetes» происходит от древнегреческого κυβερνήτης, что означает капитан, рулевой, пилот. В сокращении “K8S” цифра 8 - это восемь букв между K и S.

Таким образом, Kubernetes — это система для автоматизации развёртывания, масштабирования и управления приложениями, находящимися в контейнерах. Системы так же именуют «оркестратором контейнеров». Kubernetes имеет открытый исходный код и упрощает многие ручные процессы, связанные с развертыванием и масштабированием упакованных в контейнеры приложений. K8s работает не один. Чтобы он мог запустить службы и приложения в контейнерах, каждый из них должен быть оснащен средой выполнения контейнеров. Это может быть Docker, rkt или runc.

Основное преимущество использования Kubernetes, особенно, если необходимо оптимизировать рабочую версию приложения для облака, заключается в получении платформы для планирования и запуска контейнеров на кластерах физических или виртуальных машин. В более широком смысле это помогает полностью реализовать надежную инфраструктуру на основе контейнеров в производственных средах. Поэтому Kubernetes является идеальной платформой для размещения облачных приложений, требующих быстрого масштабирования, например, таких как потоковая передача данных в реальном времени через Apache Kafka.

При использовании контейнеров Docker, возникает проблема масштабирования, запуска и балансировки контейнеров сразу на большом количестве хостов. Kubernetes позволяет приложениям не зависеть от инфраструктуры (среды, в которой выполняются контейнеры), предоставляя простое API, к которому можно отправлять запросы. Разработчик, в декларативном стиле, сообщает API-серверу о том, что ему нужно, а Kubernetes работает с ресурсами системы, позволяя разработчику абстрагироваться от деталей реализации этой системы. Высокоуровневый API позволяет определять логическое группирование контейнеров в пулы, балансировку нагрузку, а также задавать их размещение.

В терминологии Kubernetes нодой называется машина в кластере Kubernetes. Подами (Pods) именуется группа контейнеров (либо единственный контейнер) с общими разделами, запускаемыми как единое

целое, объединенными общим linux namespace. Все контейнеры в одном поде связаны между собой сетью в рамках общего localhost. Фактически, атомарная вычислительная единица в системе Kubernetes не Docker-контейнер, а именно под.

Развёртывание (Deployment) — это абстракция Kubernetes, один из видов его подов, который позволяет нам управлять тем, что всегда присутствует в жизненном цикле приложения в том числе развертывании и поддержании нужного количества реплик в кластере. Речь идёт об управлении изменениями приложений. Если же приложение «живёт», то можно столкнуться с тем, что периодически изменяются требования к нему, расширяется его код, этот код упаковывается и разворачивается. При этом на каждом шаге данного процесса могут совершаться ошибки. Ресурс вида Deployment так же позволяет автоматизировать процесс перехода от одной версии приложения к другой. Это делается без прерывания работы системы, а если в ходе этого процесса произойдёт ошибка, у нас будет возможность быстро вернуться к предыдущей, рабочей версии приложения. Deployment. управляют наборами реплик (ReplicaSet) для непрерывного обновления подов. Допустим, вышло обновление приложения в новом контейнере. При изменении версии контейнера в шаблоне ReplicaSet, автоматического обновления не последует. При этом, если под со старой версией контейнера упадет, новый будет создан уже с новой версией. Но все работающие поды останутся на старой версии. Deployment в этой связи и требуется для управления ReplicaSet, устанавливая им стратегию обновления. При выходе новой версии контейнера необходимо заменить в текущем Deployment версию контейнера и он по заранее настроенным правилам начинает перезапуск ReplicaSet и соответственно подов в них.

Абстракция Replication Controller, обеспечивает запущенность необходимого набора подов. Replication Controller гарантирует, что определенное количество реплик пода будут запущены в любой момент времени.

Абстракция Service — отвечает за сетевое взаимодействие группы подов внутри кластера. В системе обычно существует несколько экземпляров одного микросервиса, соответственно каждый из них имеет свой IP-адрес. Количество подов может изменяться, следовательно, набор адресов также не постоянен. Другим частям системы для доступа к рассматриваемым подам нужен какой-то статичный адрес, который Service и предоставляет. Для того чтобы организовать доступ к приложению извне следует создать ресурс Kubernetes вида Service.

Существует несколько видов сервисов. Каждый последующий оборачивает предыдущий и добавляет некоторые правила маршрутизации. Создавая сервис более высокого уровня, автоматически создаются сервисы нижележащего типа. Кроме того, Kubernetes из коробки предоставляет

поддержку DNS внутри кластера, позволяя обращаться к сервису по его имени.

Для выполнения запросы к API-серверу Kubernetes используют инструмент командной строки – клиент Kubectl. Его следует установить на локальную машину и управлять несколькими кластерами с единого интерфейса.

Задачи

1. Определить материнскую (host) операционную систему (Windows, Linux, Mac OS). Установить Docker. Целесообразно использовать результаты практической работы №2.
2. Развернуть мини-кластер посредством kind, k0s или minikube (потребуется гипервизор). Определить наименование кластера.
3. Установить инструмент командной строки kubectl.
4. Убедиться, что кластер создан и в kubectl добавилась конфигурация (context) кластера, продемонстрировать существующие пространства имен (namespace) кластера.
5. Описать один под, в котором запустить контейнер с последней версией nginx. Использовать спецификацию (манифест) в формате yaml, содержащую корректную версию Kubernetes, вид создаваемого объекта (pod), наименование, образ, порт, прослушиваемый контейнером (80).
6. Создать под и продемонстрировать результат.
7. Описать манифест для нескольких подов с nginx. Использовать вид ReplicaSet, определить количество запущенных реплик самостоятельно.
8. Применить манифест для нескольких подов с nginx.
9. Проверить механизм отказоустойчивости в кластере в случае падения одного из подов, например, удалить вручную.
10. Описать манифест для управления развертыванием подов, задавая им стратегию обновления нескольких подов с nginx. Использовать вид Deployment, стратегию обновления RollingUpdate определить количество запущенных реплик самостоятельно.
11. Изменить версию контейнера с nginx через kubectl.
12. Проверить результат перезапуска ReplicaSet и подов в них, убедиться, что старый объект ReplicaSet отключил все свои поды.
13. Откатить обновление, отключить поды новой ReplicaSet и запустить старые. Вывести список ReplicaSet.
14. Подготовить материалы в отчет о выполнении практического курса занятий.

3. ОТЧЕТНЫЕ МАТЕРИАЛЫ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКИХ РАБОТ

Защита отчетов о выполнении практических работ будет проходить на заключительных практических занятиях семестра. Отчеты должны содержать результаты, подтверждающие выполнение задач практических работ. Выводы о полученных навыках, умениях, владениях, ответы на поставленные вопросы. Недопустимо включать в отчет исходные тексты программ, вместо этого следует ограничиться документацией ключевых методов, классов.

3.1. Материалы, предъявляемые к сдаче отчета

Дополнительно к отчету предъявляются следующие материалы:

- макет инфраструктуры web-сервера;
- реализованные web-сайты и их ресурсы: html-страницы, css-файлы, скрипты web-сайтов;
- распечатанный отчет.

Отсутствие любого из перечисленных материалов является основанием для переноса даты сдачи отчета.

Отчет необходимо оформить согласно следующим правилам:

- гарнитура «Times New Roman», 14pt, полуторный интервал;
- объем пояснительной записки не менее 20 страниц;
- исходный код скриптов, содержимое CSS-, HTML-файлов не должны присутствовать в пояснительной записке.

3.2. Критерии оценки результатов работы

Критерии оценки отчета о выполнении практических работ включают следующие правила.

Не принимаются к защите отчеты о выполнении практических работ, имеющие пояснительные записки, оформленные с нарушением требований п. 3.1.

Не принимаются к защите отчеты о выполнении практических работ, не соответствующие требованиям полноты материалов, предоставляемых к защите согласно п.3.1.

На повышение итоговой оценки отчета оказывают влияние следующие результаты (в порядке приоритета):

- выполнение всех задач практических работ;
- уровень качества оформления и подготовки отчета;
- количество и сложность разработанных компонентов, функционирующих на стороне сервера при выполнении отдельных заданий;

- сложность реализации архитектуры базы данных, нормализация отношений базы данных при выполнении отдельных заданий;
- применение актуальных инженерных подходов и умелое использование доступных библиотек при выполнении отдельных заданий.

На снижение итоговой оценки отчета могут оказать непосредственное влияние следующие равнозначные недостатки:

- частичное выполнение задач практических работ;
- признаки заимствований;
- избыточное комментирование исходного кода;
- трудности объяснения элементов исходных текстов (содержимого файлов *.css, *.html, *.php, *.py, *.js и др.);
- трудности при объяснении серверной архитектуры, особенностей взаимодействия с базой данных, работы web-сервера, СУБД;
- отсутствие ответов на вопросы по существу темы.

ЗАКЛЮЧЕНИЕ

В настоящих методических рекомендациях по выполнению практических работ, предусмотренных программой дисциплины «Информационное обеспечение предприятия», изложены материалы, способствующие формированию необходимых умений и владений теоретическими знаниями, полученными в рамках освоения лекционного курса. Задания практикума синхронизированы с рабочей программой дисциплины «Информационное обеспечение предприятия», подача упражнений и задач выполнена в соответствии с принципом от простого к сложному. Методические рекомендации включают элементы справочно-сопроводительного аппарата, содержат иллюстративный и табличный материал.

СПИСОК ЛИТЕРАТУРЫ

Основная литература:

1. Учебное пособие Информационное обеспечение предприятия. Часть I / Крюков Д.А. – М.: МИРЭА – Российский технологический университет, 2022. – 90 с.
2. Программирование Cloud Native. Микросервисы, Docker и Kubernetes / И.И. Портянкин, — Москва: Ridero, 2022. — 248 с.
3. HTML, CSS, Scratch, Python. Моя первая книга по программированию : руководство / С.В. Голиков, Е.В. Дубовик, Г.С. Русин, Ю.А. Иркова. — Санкт-Петербург : Наука и Техника, 2018. — 336 с.
4. Гагарин, А.Г. Практикум по разработке Web-приложений с использованием PHP и MySQL : учебное пособие / А.Г. Гагарин, А.Ф. Рогачев. — Волгоград : Волгоградский ГАУ, 2017.
5. Сейерс, Э.Х. Docker на практике / Э.Х. Сейерс, А. Милл ; перевод с английского Д. А. Беликов. — Москва : ДМК Пресс, 2020. — 516 с.
6. Самарев, Р.С. Создание простейших веб-приложений с помощью Ruby on Rails и AJAX : учебное пособие / Р.С. Самарев. — Москва : МГТУ им. Н.Э. Баумана, 2015. — 50 с.
7. Эделман, Д. Автоматизация программируемых сетей : руководство / Д. Эделман, С.С. Лоу, М. Осуолт ; перевод с английского А. В. Снастина. — Москва : ДМК Пресс, 2019. — 616 с.

Интернет-источники:

8. Портал IT-сообщества (http://habrahabr.ru);
9. Портал IT-сообщества (https://devacademy.ru)
10. Стандарты Open Web Platform (https://www.w3.org/standards/);
11. Хостинг IT-проектов (https://github.com/).

Сведения об авторе

Крюков Дмитрий Алексеевич, к.т.н., доцент, доцент кафедры корпоративных информационных систем, Институт информационных технологий.