Міністерство освіти і науки України
Державний ВНЗ «Національний гірничий університет»

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

спеціалізація *Програмне забезпечення систем*
(код і назва спеціалізації)

освітній рівень *магістр*
(назва освітнього рівня)

кваліфікація *2132.2 інженер-програміст*
(назва кваліфікації)

на тему: *Дослідження методів динамічної генерації веб-сайтів на основі мови програмування Ruby*

Виконавець:
студент 5 **курсу , групи** *121М-16-1*

(підпис) *Крюков Д.С.* (прізвище та ініціали)

Керівники	Посада, прізвище, ініціали	Оцінка	Підпис
проекту	*проф. Куваєв В.М.*		
розділів:			
Економічний	*доц. Касьяненко Л.В.*		
Рецензент	*проф. Головко В.І.*		
Нормоконтроль	*доц. Коротенко Л.М.*		

Дніпропетровськ
2018

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Національний гірничий університет»

ЗАТВЕРДЖЕНО:
завідувач кафедри

програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 ____ року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-16-1* *Крюков Д.С.*
(група) (прізвище та ініціали)

Тема дипломної роботи *Дослідження методів динамічної генерації веб-сайтів на основі мови програмування Ruby*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора Державного ВНЗ «НГУ» від *26.12.2017 р.* № *2127-л*

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – динамічний генератор веб-сайтів.

Предмет досліджень – принципи динамічної генерації вмісту та призначення веб-сайтів

Мета НДР – змістовне обґрунтування недоліків статичної генерації веб-сайтів та впровадження динамічного підходу у роботі з контентом у керованій мережі

Вихідні дані для проведення роботи: результати переддипломної практики, рішення алгоритму для програмного забезпечення з динамічної генерації веб-сайтів.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів, що очікуються, полягає у: створенні нового архітектурного підходу до фреймворків базованих на принципах динамічного маніпулювання вмістом

Практична цінність результатів полягає у: розробці програмного засобу завдяки якому стане можливим керування вмістом та функціями кожного окремого веб-сайту в згенерованій мережі

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подано у вигляді, що дозволять використовувати напрацьовані програмні рішення в реальній веб-розробці .

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Аналіз стану проблеми у літературі, профільних мережевих ресурсах та онлайн спільнотах. Корпоративних газетах та журналах. Періодичних та неперіодичних виданнях по розробці програмного забезпечення.	15.09.17-15.10.17
Вивчення літератури, теоретичний аналіз проблеми висунення гіпотези, завдань, формування структури	15.10.17-15.11.18
Розробка моделі динамічної генерації веб-сайту	15.11.17-15.12.17
Узагальнення й систематизація матеріалів, написання магістерської роботи	15.12.17-15.01.18

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки зменшенню кодової бази для утримання згенерованих веб-сайтів що в свою чергу мінімізує можливості збоїв та здешевить підтримку

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки можливості використання динамічних властивостей створених веб-сайтів у вільному розповсюджені інформації і боротьбі з державною цензурою у країнах з недемократичними режимами

7 ДОДАТКОВІ ВИМОГИ

Відповідність оформлення ДСТУ 3008 – 95. “Документація. Звіти у сфері науки і техніки. Структура і правила оформлення ”.

Завдання видав _____ *Куваєв В.М.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Крюков Д.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *11.09.2017 р.*

Термін подання дипломного проекту до ДЕК _____

Реферат

Пояснительная записка: 98 стр. , 19 рис., 3 приложения, 50 источников.

Объект исследования: динамический генератор веб-сайтов.

Цель магистерской работы: создание усовершенствованной методики динамической генерации веб-сайтов.

Методы исследования: при решении поставленной задачи использовались научные достижения в областях разработки информационных систем и программного обеспечения, рассматривались существующие разработки в области генерации веб-сайтов.

Научная новизна полученных результатов состоит в проведении анализа и выявлении недостатков традиционного подхода к статической генерации веб-сайтов, а также в создании методики динамической генерации веб-сайтов.

Практическое значение работы заключается в создании программных модулей, программного продукта, который способен работать в соответствии с методикой динамической генерации веб-сайтов.

Область применения. Разработанная методика может применяться для решения широкого спектра задач создания веб-страниц с динамическим контентом или меняющимся функциональным назначением.

Значение работы и выводы. Усовершенствованная методика позволяет создавать сети распространения контента со значительным сокращением как материальных затрат, так и временных, что подтверждается разработанным программным продуктом в данной магистерской работе.

Прогнозы по развитию исследований. Разработать универсальную методику обработки программного кода как данных для динамического изменения поведения и содержимого легковесных удаленных клиентов. Разработать комплекс программных продуктов для развертывания и управления сетью динамически генерируемых веб-сайтов.Расширить методику для применения на разных платформах и языках программирования.

В разделе «Экономика» рассчитаны трудоемкость разработки программного обеспечения, расходы на создание ПО и длительность его разработки.

Список ключевых слов: RUBY, RACK, HANAMI, JEKYLL, OCTOPRESS, HUGO, GITBOOK, HEXO, RSPEC, MVC, GIT, MULTISITING, METAPROGRAMMING, RUBY ON RAILS, SINATRA

Реферат

Пояснювальна записка: 98 стор., 19 рис., 3 додатки, 50 джерел.

Об'єкт дослідження: динамічний генератор веб-сайтів.

Мета магістерської роботи: створення вдосконаленої методики динамічної генерації веб-сайтів.

Методи дослідження: при вирішенні поставленого завдання використовувалися наукові досягнення в областях розробки інформаційних систем та програмного забезпечення, розглядалися існуючі розробки в області генерації веб-сайтів.

Наукова новизна отриманих результатів полягає в проведенні аналізу та виявленні недоліків традиційного підходу до статичної генерації веб-сайтів, а також у створенні методики динамічної генерації веб-сайтів.

Практичне значення роботи полягає в створенні програмних модулів, програмного продукту, який здатний працювати відповідно до методики динамічної генерації веб-сайтів.

Галузь використання. Розроблена методика може використовуватись для вирішення широкого спектра завдань створення веб-сторінок з динамічним контентом або змінним функціональним призначенням.

Значення роботи і висновки. Вдосконалена методика дозволяє створювати мережі розповсюдження контенту зі значним скороченням як матеріальних витрат, так і часових, що підтверджується розробленим програмним продуктом в даній магістерській роботі.

Прогнози щодо розвитку досліджень. Розробити універсальну методику обробки програмного коду як даних для динамічної зміни поведінки і вмісту легких віддалених клієнтів. Розробити комплекс програмних продуктів для розгортання та управління мережею динамічно генеруються веб-сайтів. Розширити методику для застосування на різних платформах та мовах програмування.

У розділі «Економіка» розраховані трудомісткість розробки програмного забезпечення, витрати на створення ПО та тривалість його розробки.

Список ключових слів: RUBY, RACK, HANAMI, JEKYLL, OCTOPRESS, HUGO, GITBOOK, HEXO, RSPEC, MVC, GIT, MULTISITING, METAPROGRAMMING, RUBY ON RAILS, SINATRA

Abstract

Explanatory note: 98 pages, 19 fig, 3 applications, 50 sources.

Research object: dynamic website generator.

Purpose of diploma project: creation improved methodology for the dynamic generation of Web sites.

Research methods: the scientific achievements in the areas of development of information systems and software were used to solve the task, and the existing developments in the field of web site generation were considered.

Scientific novelty of the results is to analyze and identify the shortcomings of the traditional approach to the static generation of websites, as well as to create a methodology for the dynamic generation of Web sites.

Practical importance of the work lies in the creation of software modules, a software product that is able to work in accordance with the method of dynamic generation of websites.

Scope of application. Developed methodology can be used to solve a wide range of tasks for creating web pages with dynamic content or changing functional purpose.

Meaning of the work and conclusions. Improved methodology allows creating content distribution networks with a significant reduction in both material costs and time, which is confirmed by the developed software product in this master's work.

Forecasts for the development of research. Develop a universal methodology for processing program code as data for dynamically changing the behavior and content of lightweight remote clients. Develop a set of software products for the deployment and management of a network of dynamically generated websites. Extend the methodology for application on different platforms and programming languages.

In the section "Economics", the complexity of software development, software development costs and the duration of its development are calculated.

Keyword List: RUBY, RACK, HANAMI, JEKYLL, OCTOPRESS, HUGO, GITBOOK, HEXO, RSPEC, MVC, GIT, MULTISITING, METAPROGRAMMING, RUBY ON RAILS, SINATRA

ЗМІСТ

Перелік скорочень

ПК - персональний комп'ютер

ОС - операційна система

ПЗ - програмне забезпечення

БД - база даних

СУБД - Система управління базами даних

Сайт - веб-сайт

CMS - Content Management System

JSON - JavaScript Object Notation

SEO - Search Engine Optimization

SMM - Social Media Marketing

ВСТУП

Актуальність теми дипломної роботи. У наш час великого значення набуває поширення інформації, метою для цього може бути як звичайна маркетингова діяльність так і поширення певних новин або фактів у досить розрізненому мовному чи доменному середовищі. Для цих цілей існує велика кількість інструментів починаючи від сервісів безкоштовного хостингу які дозволяють швидко розгортати сторінки і закінчуючи платними хостингами на яких може встановлюватись спеціалізоване забезпечення для керування даними. Та при певних особливих запитах на зберігання та утримання великої кількості веб-сайтів доцільніше використовувати статичні сторінки, це дозволяє зменьшити вартість хостингових послуг та підвищити загальну “міцність” системи до хакерських атак. Та у статичних сайтів досить багато інших проблем, які пов'язані з складнощами при адмініструванні та оновленні великого масиву файлів, така система вразлива до похибок у перелінковці та загальній зв'язності. Також втрачається зручність разі якщо залучені домені імена необхідно налаштувати на інший тип діяльності або якщо сутність розгорнутої мережі потребує дуже швидких змін, які потрібні власниками негайно. Саме ці проблеми вирішуються системою динамічної генерації веб-сайтів коли з одного боку ми отримуємо статичні веб-сторінки з необхідним вмістом який ми можемо швидко змінювати, з іншого така динамічна система має в собі ресурс до зміни фунцкіональності за однією командою з центрального сервера.

Об’єкт дипломної роботи – динамічний генератор веб-сайтів

Предмет дослідження – принципи ефективної динамічної генерації веб-сайтів, наповнення їх вмістом та керування мережею таких сайтів

Метою даної дипломної роботи є створення вдосконаленої методики динамічної генерації веб-сайтів, реалізація даної методики в програмному продукті який буде здатний розгортати мережу сайтів з динамічним вмістом та

функціональністю яка може бути керованою з одного єдиного центру. Реалізація основної мети передбачає вирішення наступних завдань:

1. Визначити сутність методики динамічної генерації веб-сайтів та її відмінність від інших
2. Проаналізувати стан питання генерації веб-сайтів та масовим наповненням вмістом
3. Сформулювати вимоги до програмного продукту який буде працювати відповідно до визначеної методики та реалізувати його

Обгрунтованість і достовірність наукових положень. Магістерська робота обґрунтована коректністю поставлених проблем і прийнятих припущень при описі процесів, аналізом вже готових програмних рішень та методик, порівнянням між можливостями різних програмних продуктів і засобів.

Наукова новизна отриманих результатів полягає в створенні нового архітектурного підходу до фреймворків базованих на принципах динамічного маніпулювання вмістом.

Практичне значення отриманих результатів полягає в розробці програмних модулів, програмного продукту, що дозволяють оцінити переваги динамічної генерації веб-сайтів.

Зв'язок роботи з державними програмами, планами науково-дослідних робіт. Результати дипломної роботи можуть бути використані підприємствами, фірмами, розробниками для створення мережі сайтів та масового наповнення їх контентом.

Особливий внесок магістра складається в:

• виборі методів досліджень і технологій реалізації;

• створення програмного продукту, що реалізує механізми динамічного методу генерації веб-сайтів;

• розробці теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи генерації веб-сайтів та масового наповнення контентом;

• оцінці отриманих результатів.

Апробація результатів магістерської роботи. Основні положення і результати були докладені та обговорені на студентській науковій конференції.

Структура дипломної роботи визначена метою і завданнями дослідження та складається зі вступу, трьох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Публікація даних в WEB

При створенні власного веб-сайту перед розробником постає головне питання, якими програмними продуктами користуватись для швидкої та ефективної розробки, при цьому необхідно забезпечити зручність впровадження змін вмісту в дані, багатомовність, оптимізованість для пошукових систем тощо. Існує декілька найбільш широко використовуваних прийомів для досягнення поставленої мети:

1) Використання сторонніх сервісів
2) Використання однієї з популярних CMS
3) Розробка власної CMS
4) Створення статичних веб-сторінок вручну
5) Генератори статичних веб-сторінок

Розглянемо кожен з них.Користування сторонніми сервісами може накладати додаткові витрати на обслуговування системи, та не гарантує повної можливості у збереженні даних при зміні сервісу. Популярні CMS вразливі перед масовими вразливостями і можуть за свою сутністю бути або занадто масштабними для необхідного функціоналу або містити певні хибні базові концепції у своїй архітектурі що будуть заважати цілям веб-сайту. Самостійно написані веб-додатки також значною мірою вразливі до хакерських атак та часто потребують значних затрат для своєї розробки, часто для вирішення певної задачі написання свого власного продукту є надмірними витратами чи фактором який сповільнить бізнесові цілі. Користування статичними сторінками може допомогти в тому разі якщо нам необхідний дешевий сайт-візитівка чи сторінка презентація, при цьому можливо заощадити на більш дешевих послугах хостингу, в тому числі і за рахунок відмови від використання баз даних. При цих перевагах керування статичними сторінками несе в собі деякі складності, перш за все зв'язування контенту та внесених змін між усіма складовими, які необхідно вручну коригувати після кожного оновлення. У наш

стрімкий час, коли інформація стрімко втрачає свою вартість така кропітка робота може бути успішно автоматизована. Може здатися що ідея статичних веб-сайтів у реальності так званого Інтернету 2.0 здається абсурдною, але ті самі так звані рекламні лендінги, активно використовуються маркетологами з усього світу.

Беручи лендінги як приклад ми маємо розуміти, що для них існують типові шаблони, які наповнюються певним контентом та оптимізованими під пошукові системи ключовими словами. Після публікації лендінгів на певні домени, на них купується Інтернет трафік у вигляді користувачів які в свою чергу виконують оплачувані дії, так звані “конверсіїї” приносячи прибутки своїм власникам.

Для вирішення таких типів задач гарно підходять статичні генератори веб-сайтів. По перше вони допомагають швидко наповнювати контентом типові шаблони, а по друге на відміну від звичайних статичних веб-сайтів можливо відокремити розробку дизайну від наповнення рекламними текстами та іншим. На мою думку корінний недолік статичних систем генерації веб-сайтів, відсутність гнучкості під час оновлення даних, особливо коли мова йде про великі масштаби задля потреб онлайн-маркетингу.

1.2. Landing pages

Сторінка-вітрина, також лендінг (англ. Landing page) — це завершальна сторінка воронки продаж, також — веб-сторінка, яка відкривається при натисканні на рекламне оголошення чи ланку (лінк).

«Цільова сторінка» є логічним продовженням рекламного оголошення або посилання. Часто «лендінги» пов'язані з соціальними медіа, розсилками електронною поштою або маркетинговими кампаніями пошукових двигунів (контекстною рекламою) з метою підвищення ефективності реклами. «Лендінг» може бути будь-якою сторінкою сайту або спеціально створеною окремою сторінкою. Загальна мета «лендінгу» перетворення (конверсії) відвідувачів сайту в потенційних покупців, тому її ще часто називають «приманка для клієнтів».

В залежності від маркетингової стратегії, процес перетворення, передбачає виконання користувачем певних дій, як-то: оформити замовлення на покупку, продати конкретний продукт в конкретній ситуації (розпродаж, промоакція), залишити контактну інформацію (зазвичай: адреса електронної пошти або номера телефону) підписатися на розсилку, реєстрація на сайті. інформація для інших користувачів / оголошення про цільову сторінку. віра в бренд — витратити достатньо часу на сайті, дивитися фільми, читати листівки завантажити або встановити якусь програму

Відсоток конверсії відвідувачів є показником ефективності сторінки-вітрини, в залежності від галузі він коливається в межах 2-3 відсотків. Це означає, що, наприклад, із 1000 осіб які відвідали «лендінг» - 20-30 зробили якусь із вищевказаних дій.

1.3. Статичні генератори сайтів

Останнім часом у сфері веб-розробки стало помітним сильне зміщення акцентів на користь статичних сайтів. Такі сайти простіше в обслуговуванні (ніяких баз даних, ніяких серверних сценаріїв) і більш безпечні в цілому, враховуючи, що єдина річ, яка передається на пристрої користувачів - це файли HTML, CSS і Javascript. Щоб зробити деякі типи сайтів, на кшталт блогів і сайтів з документацією, статичними, просто написати код у HTML-файлах досить складно. Також непросто підтримувати сайти з масивним контентом, особливо якщо потрібно змінити деякі незначні деталі (наприклад, дизайн). Саме в такому випадку на допомогу приходять генератори статичних сайтів. Такі генератори в основному конвертують (або компілюють) пачку різних вихідних файлів в один сайт. Це означає, що контент можна зберігати окремо від коду макета, а вміст сайту, таке як зображення, можна зберігати взагалі в іншому місці. Існує безліч генераторів статичних сайтів, можливо навіть сотні, тож розглянемо деякі найбільш популярні з них.

Jekyll - це самий широко використовуваний генератор, що включає чудову документацію, величезне співтовариство і відмінну підтримку. Навіть GitHub пропонує вбудовану підтримку Jekyll в сервісі GitHub Pages. Jekyll може похвалитися підтримкою блогів. Створювати статичні блоги на Jekyll дійсно просто. Для цього потрібні лише базові знання веб-розробки. Jekyll дозволяє створювати і використовувати різні плагіни, теги, і навіть робити свої власні конвертери для будь-якої мови розмітки, який ви хочете використовувати в Jekyll. Стандартна мова розмітки Jekyll, як і в більшості інших генераторів - Markdown. Jekyll включає плагіни для компілювання Less, Stylus, генерування хмари тегів, призначених для користувача сторінок для блогів і багато чого іншого. Jekyll заснований на движку Liquid Template від Shopify. Він працює повністю на Ruby, тому його легко встановлювати разом з взаємозалежностями, використовуючи rvm або за допомогою пакувальника. Jekyll також включає опції міграції, якщо ви хочете перенести щось з WordPress, Blogger або будь-якого іншого сайту блогів. Він без сумнівів є першим з генераторів сайтів з

точки зору кількості користувачів. І він знаходиться на стадії активної розробки.

Pelican - це генератор статичних сайтів на Python. Він відрізняється розміщенням мультимовного контенту, виділенням коду (синтаксису), а також простим генеруванням RSS і Atom Feeds. Pelican включає непоганий набір плагінів, які знаходяться в центральному репозиторії GitHub. Він підтримує три формату документів за замовчуванням: Markdown, reStructuredText і Ascii Doc.Pelican досить унікальний, оскільки він створений на Python. Він підтримує потужний движок Jinja Template Engine, заснований на пітона, що дозволяє легко створювати красиві теми і шаблони для Pelican. З точки зору підтримки міграції, Pelican пропонує підтримку WordPress і Tumblr. Замість стандартних YAML-файлів конфігурації, в Pelican для конфігурації і настройки використовується файл .py під назвою pelicanconf.py.

Metalsmith пишається тим, що він є генератором статичних сайтів на основі плагінів. Це означає, що вся логіка Metalsmith здійснюється плагінами. Яка б функція вам не була потрібна, просто додайте потрібний плагін. Величезна кількість пропонованих в Metalsmith плагінів здатне обійти будь-якого конкурента (напевно, крім Jekyll і Docpad). Це означає, що Metalsmith можна використовувати як щось більше, ніж просто генератор статичних сайтів. Іншими словами «Так як все складається з плагінів, бібліотека насправді являє собою просто абстракцію для управління директорією файлів». В результаті це дає те, що ви легко використовуєте Metalsmith як інструмент для роботи з проектом, генератор електронних книг, інструмент для побудови технічної документації та ін. (Це всього кілька прикладів, продемонстрованих на сайті Metalsmith).

Harp включає вбудовану попередню обробку для Jade, Markdown, LESS, Sass, Coffeescript, EjS і Stylus без будь-яких додаткових налаштувань. Він також дозволяє використовувати макети / часткові таблиці форм з Jade і EjS, для чого в інших генераторах статичних сайтів потрібен спеціальний плагін.

Harp побудований на Node.Js і може працювати пліч-о-пліч з Harp Platform, що дозволяє створювати веб-сторінки з папки на Dropbox. Harp також може компілювати сторінки для використання на сторінках GitHub, а також PhoneGap і Heroku.

Docpad - це динамічний генератор статичних сайтів. Він пропонує більш великі можливості, ніж інші генератори статичних сайтів, пропонуючи такі функції, як запити бази даних через движок запитів, імпорт сторінок з зовнішніх баз даних, повторне відображення веб-сторінки при кожному запиті. Docpad володіє вбудованою підтримкою препроцесорів, таких як Coffeescript, Stylus і LESS, і використовує плагіни для підтримки двигунів шаблонів, препроцесорів і мов розмітки, тому ви можете вибирати будь-які бажані комбінації, використовуючи необхідний плагін. Docpad також підтримує імпорт сторінок з зовнішніх джерел, таких як Tumblr, GitHub і Dropbox, через плагіни. Docpad - це багатофункціональна платформа з величезною кількістю плагінів і відмінною документацією. Для зв'язку з сервером він використовує Node.js, на якому і побудований.

Hexo - це легка оболонка для статичних блогів, яка відрізняється величезною швидкістю генерування сайтів. Hexo відмінно підходить для блогерів з великою кількістю контенту, яким потрібен простий генератор статичних сайтів. Він пропонує зручну функцію міграції з інших платформ блогів, таких як WordPress, Joomla, Jekyll, Octopress і RSS. А що найважливіше в Hexo - це те, що в ньому можна використовувати більшість плагінів, створених для Octopress (і, отже, плагіни, створені для Jekyll з дуже незначними змінами).Hexo підтримує Markdown, YAML для створення титульної сторінки і налаштування. І, не втрачаючи своєї характерної швидкості, Hexo дозволяє розгортатися на таких сайтах як GitHub, Heroku і Rsync за допомогою всього однієї команди.

Hugo - це генератор статичних сайтів загального призначення з відмінними універсальними функціями, такими як підтримка шаблонів і компонентів, розбивка на сторінки і «таксономія», яка спочатку є унікальною

системою категоризації, успадкованої з Hugo. Це означає, що ви можете розбивати пости на класи не тільки на підставі тегів, але також будь-яким іншим способом на ваш розсуд, наприклад за категоріями або серіями, починаючи з титульної сторінки. Hugo підтримує три типи файлів даних - YAML, JSON і TOML, і дозволяє вибирати, з чим вам зручніше працювати. Замість плагінів в Hugo використовуються «короткі коди», які дозволяють використовувати багатий контент всередині Markdown. Щоб дізнатися, як це працює, ознайомтеся з цією статтею. Hugo написаний на мові програмування Go і пропонує окремі файли установки для різних платформ на своїй сторінці GitHub.

Brunch швидше спрямований на веб-додатки в HTML5, ніж на блоги та веб-сайти, але все одно це дуже простий у використанні і швидкий генератор статичних сайтів. Він не тільки компілює весь ваш код і скрипти, але також може автоматично зменшувати (мінімізувати) ваш код і стискати зображення. Brunch включає цілий ряд плагінів, які ви можете використовувати для настройки генератора відповідно до своїх потреб. Brunch пропонує «скелети», які в основному є шаблонами для створення сайту (або веб-додатки). Він забезпечує мало не найвищу швидкість компіляції, просто тому що Brunch кешує всі незмінні частини вашого проекту і компілює тільки ті файли, які змінилися. Brunch побудований на Node.js.

1.4. Системи керування вмістом

Система керування вмістом (англ. Content Management System, CMS) програмне забезпечення для організації веб-сайтів чи інших інформаційних ресурсів в Інтернеті чи окремих комп'ютерних мережах. Існують сотні, а може, навіть й тисячі доступних CMS — систем. Завдяки їх функціональності ці системи можна використовувати в різних компаніях. Незважаючи на широкий вибір інструментальних та технічних засобів, наявних в CMS, існують загальні для більшості типів систем характеристики.

Розрізняють наступні типи CMS:

Web content management systems для управління веб-сайтами (наприклад, енциклопедіям, онлайн-виданнями, блогами, форумами, корпоративними чи персональними веб-сторінками та ін.)

Транзакційні CMS для забезпечення транзакцій у електронній комерції.

Інтегровані CMS для роботи з документацією на підприємствах.

Електронні бібліотеки (Digital Asset Management) для забезпечення циклу життя файлів електронних медіа (відео, графіки, презентації тощо).

Системи для забезпечення циклу життя документації (інструкції, довідники, описи).

Освітні CMS — системи для організації Інтернет курсів та відповідного циклу життя документації

Платформенні CMS (Platform Content Management Systems) підтримують автоматизацію роботу з комп'ютерними файлами, папками, програмами у визначеному програмному середовищі.

Корпоративні CMS (Enterprise content management systems) з різноплановим пристосуванням для потреб підприємницької діяльності. Підтримують цикл життя внутрішньої і зовнішньої документації.

Також існують різні типи роботи CMS:

Генерація сторінок за запитом. Системи такого типу працюють на основі зв'язки «модуль редагування - база даних - модуль представлення». Модуль представлення генерує сторінку з контентом при запиті на нього на основі

інформації з бази даних. Інформація в БД змінюється за допомогою модуля редагування. Сторінки заново створюються сервером при кожному запиті, а це створює навантаження на сервер. Але це навантаження може бути багатократно зменшене при використанні методів кешування, які є в сучасних веб-серверах.

Генерація сторінок при редагуванні. Системи цього типу при редагуванні сторінок вносять зміну у вміст сайту та створюють набір статичних сторінок. При такому способі втрачається інтерактивність між відвідувачами сайтів та контентом даного сайту.

Змішаний тип. Як зрозуміло із назви, цей тип поєднує в собі переваги перших двох. Може бути реалізований шляхом кешування — модуль представлення генерує сторінку один раз, надалі вона через деякий час буде в декілька разів швидше завантажуватися із кешу. Кеш може оновлюватись як автоматично, через деякий час чи при внесенні змін у певні розділи сайту, так і вручну за командою адміністратора. Другий підхід — збереження певних інформаційних блоків на етапі редагування сайту і збирання сторінок з цих блоків при запиті відповідної сторінки користувачем.

Для розуміння різниці між статичними генераторами веб-сайтів та CMS розглянемо особливості найбільш популярних.

Joomla - важливою особливістю системи є мінімальний набір інструментів при початковій установці, який доповнюється в міру необхідності.

Функціональність можна збільшувати за допомогою додаткових розширень (компонентів, модулів і плагінів).

Є модуль безпеки для багаторівневої аутентифікації користувачів та адміністраторів (використовується власний алгоритм аутентифікації і «ведення» сесій).

Система шаблонів дозволяє легко змінювати зовнішній вигляд сайту або створити свій унікальний. У мережі існує величезний вибір готових шаблонів, як платних, так і безкоштовних.

Передбачені схеми розташування модулів, включаючи лівий, правий, центральний і будь-яке інше довільне положення блоку. При бажанні вміст модуля можна включити в вміст матеріалу. Вбудована багатомовність.

Розширена підтримка баз даних. Реалізована підтримка Microsoft SQL Server, а з версії 3.0 – PostgreSQL

Drupal - популярна вільна модульна система керування вмістом з відкритим сирцевим кодом, написана на мові програмування PHP.

Drupal може працювати у таких популярних системах як Windows, Mac OS X, Linux, власне, на будь-якій платформі, яка підтримує роботу веб-сервера Apache, Nginx, Lighttpd або Microsoft IIS; також потрібна наявність системи керування базами даних MySQL/MariaDB, PostgreSQL 8.3, SQLite чи інші комерційні.

У дистрибутив системи входить набір модулів, що дають наступні можливості:

- збір інформаційних стрічок (RSS, RDF, Atom);
- ведення блогів, підшивань і форумів;
- створення форм для відправки повідомлень;
- локалізація системи;
- перейменування посилань (призначення посиланням зрозумілих і зручних псевдонімів);
- проведення опитувань;
- призначені для користувача профілі, що налаштовуються;
- пошук за змістом (за зміст вважається і повідомлення на форумах, і сторінки, і будь-які інші призначені елементи);
- ведення журналу статистики (відвідуваності);
- таксономія (впорядковування матеріалу за категоріями) — дуже «цінна» можливість;
- формування сторінок з матеріалами в різних формах і форматах подання та інші.

WordPress - проста у встановленні та використанні система керування вмістом з відкритим кодом, яка широко використовується для створення веб-сайтів. Сфера застосування — від блогів до складних веб-сайтів. Вбудована система тем і плагінів в поєднанні з вдалою архітектурою дозволяє конструювати на основі WordPress практично будь-які веб-проекти. Написана мовою програмування PHP з використанням бази даних MySQL. Сирцевий код поширюється на умовах ліцензії GNU General Public License.Окрім цього Розробники Wordpress дали можливість користувачам створювати власні плагіни. Всі файли плагінів розміщуються в теці wp-content/plugins. Його головний файл повинен бути написаний на мові PHP. Плагін також може складатись з декількох файлів, якщо вони під'єднані до головного файлу (наприклад за допомогою функції include). Якщо ж до нього треба приєднати CSS, JavaScript або інші зовнішні файли.

Дизайн, управління системою та інші можливості:

- простота встановлення, простота налаштувань;
- підтримка веб-стандартів (XHTML, CSS);
- модулі для підключення (плагіни) з унікально простою системою їх взаємодії з кодом; можливість автоматичного встановлення та оновлення версії безпосередньо з панелі адміністратора;
- підтримка так званих «тем», з допомогою яких легко змінюється як зовнішній вигляд, так і способи виведення даних;
- можливість редагувати шаблони одразу в панелі адміністратора;
- «теми» реалізовані як набори файлів-шаблонів на PHP (у HTML-розмітку вставляються PHP-мітки);
- багато бібліотек «тем» і «плагінів»;
- потенціал архітектури дозволяє легко реалізовувати складні рішення;
- SEO-оптимізована система;
- наявність українського перекладу.

Публікація та редагування:

- миттєва публікація;
- підтримка RSS, Atom, trackback, pingback;
- наявність ЛЗУ (людино-зрозумілий URL);
- редагування WYSIWYG-редактором з можливістю вставлення форматованого тексту (наприклад з програми Microsoft Word) або редагування за допомогою HTML-розмітки.

Контент:

- наперед заплановані публікації;
- багатосторінкові записи;
- прикріплення файлів та зображень до записів;
- можливість створення статичних сторінок;
- можливість створення свого типу контенту у власних темах;
- категорії, теги, коментування тощо.

Вказані особливості перелічених CMS дозволяють їм займати значне положення на ринку всіх встановлених веб-сайтів в мережі, та задля спрощення роботи з веб-сайтами та підвищення рівня безпеки вони не завжди можуть бути використані.

1.5. Мультисайтинг

Мультисайтинг (багатосайтовість) можна визначити як можливість використовувати файли скрипту для різних сайтів. Одним з найбільш поширених прикладів мультисайтинга може бути використання загальної бази даних користувачів на декількох сайтах. Мультисайтинг це спосіб використовувати енергію відвідувачів до кінця. На звичайних сайтах відвідувачі йдуть на інші сайти Інтернету. При мультисайтингу у відвідувача є можливість продовжити серфінг на одному з сайтів зв'язки. З огляду на, що всі сайти зроблені на одному скрипті і влаштовані приблизно однаково, відвідувач відчуває більше комфорту при серфінгу/шопінгу і йому легше зробити покупку або іншу оплачуваний дію.

Розрізняють 2 види мультисайтингу:

Мультисайтинг із загальним скриптом. Багато незалежних сайтів використовують один скрипт.

Мультисайтинг з загальними таблицями. Багато сайтів частково використовують однакові таблиці в базі, наприклад, дані про користувачів.

Варіанти загального скрипту:

Якщо сайтів не багато (2-5), то можна завести окрему папку на сервері, куди завантажувати еталонну версію скрипта і всіх модулів.

Мультисайтинг із загальним двигуном на бекап модулі (програма, яка робить backup, резервну копію). При оновленні скрипту переносити вручну або за допомогою модуля еталонну версію по локальних папках. Звідки вони будуть заливатися на сайт при черговому оновленні сайту.

Мультисайтинг із загальним двигуном на FTP-синхронізаторі. Якщо ми не збираємося налагоджувати локально сайти з використанням останньої версії скрипту, то можна використовувати мультисайтинг із загальним движком ще простіше.Береться FTP-синхронізатори з його допомогою заливається вміст еталонної папки безпосередньо на сайти. Мультисайтинг із загальним скриптом засобами сервера. Якщо частина сайтів розташована на одному сервері, то з'являється можливість перенести папку з еталонним

движком на сервер. Фізично папка з скриптом буде одна для всіх сайтів. Сервер сам буде "розмножувати" цю папку для всіх сайтів. Варіанти з загальними таблицями:

Якщо кожен сайт зберігає таблиці у своїй незалежній базі, то сайти повністю незалежні (з точки зору відвідувачів).

Сайти об'єднують свою базу користувачів, але зберігають незалежність в іншому. Тут таблиці, що відповідають за користувачів, у сайтів загальні, а все інше різне.

Сайти об'єднують користувачів і частина вмісту. Наприклад, у різних сайтів загальні користувачі і загальний форум. Статті різні на кожному сайті. Пошуковики не люблять, коли один і той же вміст з'являється в різних місцях. Їм незрозуміло, що є першоджерелом. В результаті буде показано вміст тільки на одному сайті. Причому вибір може бути зроблений випадково. Так що відвідувач запросто може потрапити на найслабший сайт з пов'язаних в мультисайтингу. У найгіршому випадку пошуковики можуть покарати інші сайти за спробу обману пошукових систем. Відвідувачам теж незручно мати справу з частковим зеркалюванням сайту. Посилання на дзеркальне вміст показуються в його браузері ще не переглянутих.

Сайти об'єднуються повністю. Виходять "дзеркала". Раніше сайти дзеркала були поширені як спосіб боротьби з поганим хостингом. Якщо не було видно сайту на одному місці, то відвідувач міг знайти його на іншому. Зараз дзеркала вижили тільки для файлових архівів, в основному на безкоштовних сховищах. Комерційний сайт не повинен віддзеркалюватись. Пошуковики можуть показувати одні посилання на такий сайт і www, а інші без. Відвідувач зайде по посиланню з www, перегляне сайт і побачить на форумі посилання без www. Її дав той, хто зайшов за адресою без www. Але браузер може показати її як ще не відвідану. І відвідувачеві піде вдруге дивитися ту ж саму сторінку.

Якщо проводити кордон об'єднання таблиць, то найоптимальніше об'єднати зареєстрованих користувачів, але не замахуватися на об'єднання

вмісту, тобто використовувати другий варіант зазначений вище. Сайти зберігають незалежний вміст з точки зору пошукових систем і відвідувачів. Але при цьому відвідувачі вільно переходять з одного сайту на інший зі збереженням свого "обличчя" і купівельної "історії" .

При мультисайтингу на загальних таблицях зареєстрований на якомусь сайті користувач автоматично отримує можливість скористатися тим же логіном і паролем при вході на інші сайти зі зв'язки. Має також сенс об'єднати і дані профілю. Якщо сайти в мультисайтинговій зв'язці працюють незалежно, то в такому об'єднанні немає сенсу. Користувачі не знають, що десь у світі є інші сайти, на яких можна скористатися тим же логіном/паролем треба в блоці логіна на кожному сайті зі зв'язки повісити оголошення про це і дати список сайтів зі зв'язки. Якщо відвідувач вже зареєстрований на якомусь із сайтів зв'язки, то це спонукає його використовувати наявний логін/пароль.

Всі інші з цікавістю можуть пройти по посиланнях, щоб подивитися, що там цікавого є. Крім об'єднання призначених для користувача логінів/паролів можна об'єднати між сайтами і базу покупців. Якщо користувач зробив покупку на одному з сайтів, то на іншому сайті він може зробити покупку в спрощеному варіанті. Всі його адреси і координати підставити з анкети, яку він заповнював при своїй першій покупці на іншому сайті. Створюючи зв'язки сайтів із загальною спрямованістю можна зберегти користувачів всередині власної мережі отримуючи множинну вигоду з кожного користувача відвідування і кліка. Залучення нового клієнта в мережу буде за визначенням коштувати завжди дорожче ніж закільцювати переходи вже прийшов клієнта, варто звернути увагу на технологію мультисайтингу як на ефективний спосіб розширювати свою присутність на ринках і завойовувати окремі ніші. Найбільше в даному контексті дана методика підходить для торгових або медійних мереж які спроможні пропонувати користувачеві нові і нові гілки переходів на інші сайти.

1.6. Шаблон MVC – головні положення

Шаблон проектування Model-View-Controller (далі просто MVC) ліг в основу архітектурного рішення першої середовища програмування з графічним інтерфейсом користувача - Smalltalk-80. Вперше MVC описав ще в 1978 році норвежець Трюгве Рінскауг, який працював деякий час в лабораторії Xerox PARC. Реалізацію шаблону в Smalltalk-80 Стів Бурбек.

Шаблон проектування MVC передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно. Модель (Model) надає дані предметної області виду і реагує на команди контролера, змінюючи свій стан. Вид (View) відповідає за відображення даних предметної області (моделі) користувачеві, реагуючи на зміни моделі. Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін (Рис 1.1).

Розглянемо простий приклад MVC. Модель може являти собою об'єкт, який реалізує перемикач. У найпростішому випадку дана модель характеризується станом: вимкнений або включений - і, крім того, дозволяє змінювати його - об'єкт моделі має метод зміни стану: вимкнути і включити. Вид відображає на дисплеї стан перемикача за допомогою певної текстової або графічної форми. Наприклад, уявлення може відображати текстову мітку, яка при зміні стану моделі перемикача відобразить відповідний текст: «вимкнений» або «включений». Крім відображення уявлення дозволяє користувачеві змінювати стан перемикача за допомогою графічних примітивів, наприклад, двох кнопок з написами: «Включити» і «Вимкнути». Вид вміє тільки відображати стан моделі перемикача, для зміни виду звертається до контролера. Контролер являє собою об'єкт, який в нашому випадку вміє тільки змінювати стан моделі перемикача. Запропоноване програмістами Smalltalk-80 рішення виявилося настільки ефективним, що по закінченні вже майже 30 років з моменту своєї появи шаблон проектування MVC досі є стандартом настільних і Інтернет-додатків. В цьому легко переконатися - досить розглянути, наскільки

MVC представлений в популярних платформах програмування.MVC задає не тільки правила поділу додатки на окремі компоненти, скільки правила їх взаємодії. У той час як уявлення і контролер залежать від моделі, модель не залежить ні від уявлення, ні від контролера. Це ключова особливість поділу, яка дозволяє працювати з моделлю, а значить, і з бізнес-логікою програми, незалежно від візуального представлення.

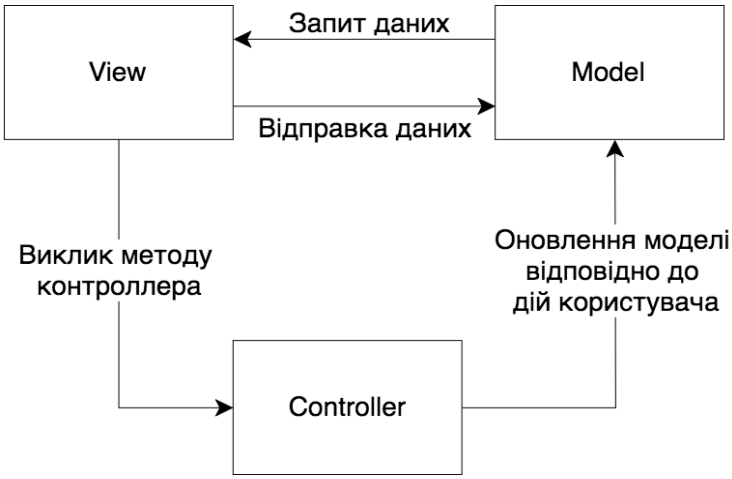

Рис.1.1 Схема шаблону MVC

Вид безпосередньо управляє відображенням інформації користувачеві. Сама ця інформація в термінах шаблону проектування MVC називається моделлю. Контролер співпрацює з видом, гарантуючи інтерпретацію дій користувача над даними, знову ж таки - відображенням моделі. Тому уявлення і контролер можуть бути обмежені типом даних (моделлю), з яким вони працюють. Це обмеження дозволить використовувати уявлення і контролер спільно для роботи моделлю.Навпаки, модель не повинна мати обмежень. Обмеження на тип об'єкту, який може функціонувати в якості моделі, в свою чергу обмежить рамки використання тріади MVC. Необхідно, щоб будь-який об'єкт міг бути моделлю. Наприклад, число з плаваючою точкою може виступити моделлю для відображення температури.З іншого боку, модель може бути обмежена типом даних, який вона представляє, і який успадковується від суперкласу.

В описі оригінальної реалізації MVC в Smalltalk згадується про пасивної і активної моделі. Пасивна модель не знала про існування уявлення, контролера, і навіть про свою участь в MVC-тріаді. Контроллер відстежує зміни моделі і оповіщає уявлення.

При цьому або контролер передає поданням інформацію про зміни, або уявлення самостійно вибирає дані з моделі. Більш витонченим рішенням є активна модель.Активність моделі проявляється в її праві самостійно оповістити уявлення про зміну свого стану. Щоб не порушити основна вимога MVC про незалежність моделі від уявлення і контролера, механізм оповіщення реалізується на основі шаблону Observer.

У завдання контролера входить перетворення дій користувача в виконання певної операції над моделлю. Найбільш підходящим рішенням є реалізація шаблону проектування Command.Шаблон проектування Command (Команда) (він же Action (Дія) або Transaction (Транзакція)) використовується, якщо необхідно надіслати об'єкту запит, не знаючи про те, виконання якої операції запрошено, і хто буде її одержувачем. Рішення полягає в тому, що запит інкапсулює об'єкт Команда, який задає інтерфейс виконання операції, визначаючи тим самим зв'язок між одержувачем і ініційованим дією. Ініціатор створює об'єкт Конкретна Команда і відправляє його в запиті на виконання, клієнт приймає запит, встановлює одержувача запиту і перетворює об'єкт Конкретна Команда в набір дій над одержувачем. Шаблон проектування Command розриває зв'язок між об'єктом, який ініціює операції, і об'єктом, що має інформацію про те, як операції виконувати. Крім того, створюється об'єкт, який можна розширювати через наслідування. Згадайте представників керівних посад. З одного боку, начальник бажає керувати всіма підлеглими за допомогою простих зрозумілих йому вимог. Але з іншого боку, він хотів би якомога менше знати, по-перше, про конкретних виконавців своїх вказівок, а по-друге, про те, які саме дії будуть робити співробітники, щоб виконати завдання.

1.7. Мова програмування Ruby

Ruby (Рис.1.2) інтерпретована мовою. Це означає, що виконується безпосередньо вихідний код програми, а не спеціально підготовлені файли з наборами машинних команд, як у випадках використання компілятора. Перевагою інтерпретатора є простота використання і машиннонезалежність, так як на виході немає машинного коду.

Рис.1.2 Логотип Ruby

Таким чином, програма, створена в одній операційній системі, буде без змін працювати в інший. Відзначимо, однак, що програма, яка використовує для свого виконання інтерпретатор, як правило, працює значно повільніше (програма на Ruby зажадає приблизно в 300 разів більше часу на виконання, ніж аналогічна програма на мові C).

Ruby - це ретельно збалансована мова. Її творець Юкіхіро Мацумото відомий як "Matz", об'єднав частини його улюблених мов (Perl, Smalltalk, Eiffel, Ada і Lisp) щоб сформувати нову мову, в якій парадигма функціонального програмування збалансована принципами імперативного програмування. Він часто повторював, що "намагається зробити Ruby природною, але не простою" мовою, яка відображає життя.

Спочатку Matz розглядав інші мови в пошуках ідеального синтаксису. Згадуючи свої дослідження, він говорив: "Мені потрібен був скриптова мова,

який був би більш потужним, ніж Perl, і більш об'єктно-орієнтованим, ніж Python.". Рубі вбирає в себе їх кращі сторони.

В Ruby все - об'єкт. Для кожної частинки інформації або коду можуть бути визначені власні властивості і дії. В об'єктно-орієнтованому програмуванні властивості називаються змінними об'єкта, а дії - методами. Найчистіше об'єктно-орієнтований підхід Ruby може бути продемонстрований парою рядків коду, в яких проводиться дію над числом, і проводиться вивід рядка.

```
5.times { print "National Mining University it’s my Alma Mater!" }
```

У багатьох мовах числа та інші примітивні типи даних не є об'єктами. Ruby під впливом мови Smalltalk дозволяє задати методи і змінні об'єкта всім типам даних. Це спрощує використання Ruby, так як правила застосовні до об'єктів - застосовні до всього Ruby.

Ruby дуже гнучка мова, яка дозволяє користувачам вільно міняти певні частини. Основні частини Ruby можуть бути видалені або перевизначені за бажанням. А до існуючі частини можна модифікувати.

Ruby намагається ні в чому не обмежувати користувача.Наприклад, додавання виконується операцією плюс (+). Але, якщо ви хочете використовувати для цього більш читане слово plus - ви можете додати такий метод прямо в Numeric, внутрішній клас мови Ruby.

```
class Numeric
  def plus(x)
    self.+(x)
  end
end
y = 5.plus 6 # y тепер 11
```

Оператори в Ruby - синтаксичний цукор для методів. Ви також можете перевизначити їх. Блоки, по-справжньому виразна конструкція

Блоки в Ruby також є відмінним джерелом гнучкості. Програміст може додати замикання до будь-якого методу, описуючи, як цей метод повинен діяти.

Замикання називається блок і є однією з найбільш популярних конструкцій для тих, хто прийшов у світ Ruby зі світу імперативних мов програмування, таких як PHP або Visual Basic. Створення блоків було натхненне функціональними мовами програмування. Matz казав, "замиканнями в Ruby я хотів віддати данину поваги культурі мови Lisp."

```
search_engines =
%w[Google Yahoo MSN].map do |engine|
  "http://www." + engine.downcase + ".com"
end
```

У коді вище блок описаний всередині конструкції do ... end. Метод map застосовує блок коду до представленого списку слів. Багато інші методи в Ruby залишають шлях, відкритий для програміста, щоб той написав власний блок коду, детально говорить методу, що той повинен зробити.

На відміну від багатьох об'єктно-орієнтованих мов, Ruby навмисно надає лише одиночне спадкоємство. Але Ruby також надає концепцію модулів (званих Категоріями в Objective-C).

Класи можуть вільно вмішувати модуль і отримувати все його методи. Наприклад, будь-який клас, який реалізує метод each, може підмішати модуль Enumerable, який додасть купу методів використовують each для створення циклів.

```
class MyArray
  include Enumerable
end
```

В основному, рубісти знаходять це більш прозорим, ніж множинне спадкування, яке може бути досить складним і мати будь-які ограніченія.Так як в Ruby часто пунктуація зустрічається досить рідко і зазвичай використовуються англійські слова в якості ключових, деякі знаки пунктуації використовуються для прикраси Ruby . Ruby не потребує оголошенні змінних. У ньому використовуються прості конвенції іменування, для того щоб розділити області видимості змінних var може бути локальною змінною @var

змінна об'єкта $ var глобальна змінна.Дана символіка підвищує читабельність, дозволяючи програмісту легко ідентифікувати роль кожної змінної. Це також дозволяє не використовувати стомлююче self. для кожного об'єкта.Ruby сповнений іншими особливостями і конструкціями, і ось деякі з них:

- В Ruby є конструкції для обробки винятків, як в Java або Python, які дозволяють простіше працювати з помилками.
- В Ruby представлений справжній mark-and-sweep (познач і відчистити) збирач сміття для всіх Ruby об'єктів. Не потрібно вручну відстежувати кількість посилань в сторонніх бібліотеках. Як говорить Matz, "Це корисніше для вашого здоров'я."
- Писати розширення на C в Ruby простіше ніж в Perl або Python за допомогою дуже елегантного API для виклику Ruby з C. Він включає в себе виклики для вбудовування Ruby в програмне забезпечення, щоб використовувати його як скриптова мова. Також доступний інтерфейс SWIG.
- Ruby може довантажувати сторонні бібліотеки динамічно, якщо дозволяє операційна система.
- В Ruby реалізовані незалежні від операційної системи потоки. Таким чином, на будь-яких платформах, де ви запускаєте Ruby, ви також маєте можливість використовувати багатопоточність, не залежно від того, чи підтримує дана система потоки чи ні. Ви можете скористатися наявними можливостями багатопотоковості навіть в MS-DOS!
- Ruby відрізняється високою переносимістю: він був розроблений здебільшого на GNU / Linux, але працює на багатьох типах UNIX, Mac OS X, Windows 95/98 / Me / NT / 2000 / XP / Vista / 8, DOS, BeOS, OS / 2 , і так далі.

1.8. Готові рішення у світі Ruby

1.8.1. Ruby On Rails

Ruby on Rails є середовищем, що полегшує розробку, розгортання і обслуговування веб-додатків. За час, що минув з її початкового випуску, Rails пройшла шлях від маловідомої технології до феномену світового масштабу і, що більш важливо, стала саме тим середовищем, яку вибирають, щоб створювати так звані додатки Web 2.0.

Rails добре прижилася з самого початку. Велика кількість розробників було незадоволена тими технологіями, які застосовувалися ними для створення веб-додатків. І справа, напевно, не в тому, що саме вони використовували - Java, PHP або .NET, - у них накопичувалося відчуття зайвої трудомісткості їх роботи. А потім в один момент прийшла Rails, з якою працювати стало набагато простіше. Але сама по собі простота не означає спрощеність. Йдеться про професійні розробників, що створюють по-справжньому затребувані у всьому світі веб-сайти. Їм хочеться бачити створені ними програми що витримали випробування часом - спроектованими і розробленими з використанням сучасних, професійних технологій. Тому розробники зайнялися Rails всерйоз і виявили, що вона є не тільки інструментом для розробки веб-сайтів.

Наприклад, всі Rails-додатки виконуються з використанням архітектури Модель-Вид-Контролер (Model-View-Controller, MVC). Звична Java-розробникам середовище виконання, наприклад Tapestry або Struts, теж заснована на MVC. Але Rails йде в використанні MVC ще далі: при веденні розробки в Rails ви починаєте вже з ефектів у програмному забезпеченні, в якому є місце для кожної частини коду, і всі частини вашого застосування стандартним чином взаємодіють один з одним Професійно програмісти пишуть тести. І Rails знову вносить свою лепту. Всі Rails-додатки мають вбудовану тестування. У міру додавання до програмного коду будь-якої функціональної можливості Rails автоматично створює програмні заглушки тестів, призначені для її тестування. Це середовище полегшує тестування своїх додатків,

стимулюючи тим самим розробників до цього заняття. Rails використовує всі можливості Ruby, будучи його оригінальним розширенням, що полегшує життя програмістів. Програми стають коротшими, читаються легше. Це також дозволяє нам виконувати ті завдання, які інакше виконувалися б в вихідному коді зовнішніх файлів конфігурації. Це полегшує розуміння того, що відбувається. Наступний код визначає для проекту модель класу. Зараз не варто вдаватися в деталі цього коду - краще просто подумати про те, як багато інформації було висловлено в кількох рядках програми.

```
class Project < ActiveRecord::Base
  belongs_to :portfolio
  has_one :project_manager
  has_many :milestones
  has_many :deliverables, through: milestones
  validates :name, :description, presence: true
  validates :non_disclosure_agreement, acceptance: true
  validates :short_name, uniqueness: true
end
```

Вкоротити код Rails і зробити його більш читабельним дозволяють дві інші філософські основи цього середовища: DRY і превалювання угоди над конфігурацією. DRY означає «do not repeat yourself», тобто «ніколи не повторюватися»: кожна частинка знань в системі повинна бути виражена тільки в одному місці. Щоб втілити все це в життя, Rails користується всією ефективністю мови Ruby. У Rails-додатках можна побачити лише малу частку повторень, то, що потрібно сказати, йдеться тільки в одному місці, яке часто пропонується угодами про MVC-архітектурі, і далі про це можна вже не турбуватися, система дуже продумана в цьому плані. Для програмістів, які звикли працювати в інших середовищах веб-розробки, де проста зміна може змусити їх вносити в код програми півдюжини, а то і більше правок, це було відкриттям .Превалірованіе угоди над конфігурацією є не менш важливим принципом. Він означає, що в Rails практично для кожного аспекту, який

зв'язує в єдине ціле ваше додаток, є раціональні умовчання. Дотримуйтесь угодам, і тоді ви зможете написати Rails-додаток, використовуючи менше коду, ніж в звичайному веб-додатку, написаному на Java і використовує XML-конфігурацію. Якщо потрібно переписати угоди, Rails полегшує і це завдання.

Розробники, які переходять на Rails, помічають ще одну особливість. Rails не грає в догонялки зі стали де-факто новими стандартами: навпаки, вона допомагає їх визначати. До того ж Rails полегшує розробникам інтегрування в їх код таких функцій, як інтерфейси AJAX і RESTful, оскільки їх підтримка вже вбудована в Rails. (Якщо ви не знайомі з інтерфейсами AJAX і REST, не варто турбуватися, трохи пізніше ми пояснимо вам, що це таке.) Розробники стурбовані також розгортанням своїх продуктів. І тут виявляється, що з Rails можна поширювати вдалу версію свого додатка на будь-яку кількість серверів всього лише однією командою (і так само легко повертати все назад, якщо версія виявиться не зовсім вдалою) .Rails була виділена з реального комерційного застосування. Виявилося, що найкращим способом створення середовища є визначення основних складових конкретного додатка, а потім занесення їх до загального фонду коду. При розробці Rails-додатки в вашому розпорядженні з самого початку вже є половина по-справжньому гарного додатки .Але у Rails ще є дещо таке, що важко піддається опису. Вона якимось незбагненним чином створює упевненість в правильному виборі.

Що являє собою MVC в RoR? MVC - це патерн архітектури додатку, чітко розмежує три його компонента (Рис. 1.3):

Model (Модель) є «суттю» додатка і відповідає за безпосередні алгоритми, розрахунки тощо внутрішній устрій додатки. Також надає лінк до сховища даних.

View (Вид) призначений для виведення даних, наданих Моделлю. Це єдина частина MVC, яка безпосередньо контактує з користувачем.

Controller (Контролер) отримує дані від користувача і передає їх в Модель.Он отримує повідомлення від Моделі і передає в Представлення.

Виходячи з цього RoR використовує три компоненти:

- Active Record
- Action View
- Action Controller

Active Record - це Модель в RoR. Модель зберігає дані і надає базу для роботи з даними. Крім цього Active Record також є ORM фреймворком. ORM значить Object-relational mapping (Об'єктно-реляційна проекція).

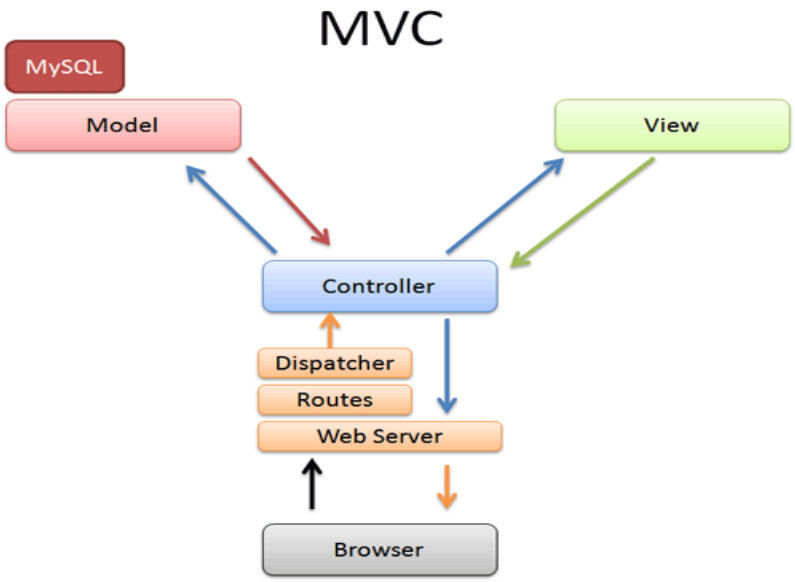

Рис. 1.3 Схема MVC в Ruby on Rails

Власне Active Record робить такі речі: Проекція таблиці на клас. Кожна таблиця проектується на один або кілька класів за принципом convention over configuration (угода вище конфігурації). Одне з таких угод - ім'я таблиці повинно бути у множині, а назва класу - в єдиному. Атрибути таблиці нальоту проектуються в атрибути примірника Рубі. Після того, як всі проекції зроблені, кожен об'єкт ORM класу представляє певну рядок таблиці, з якої клас був спроектований.

Ви можете підключитися до бази даних, використовуючи API, що надається Active Record, який створює необхідний вам запит безпосередньо в движок БД за допомогою адаптерів. У Active Record є адаптери для MySQL,

Postgres, MS SQLServer, DB2, і SQLite. Необхідно лише записати параметри доступу до БД у файлі database.yml.

Операції CRUD. Це операції create (створення), retrieve (отримання), update (оновлення) і delete (видалення) над таблицею. Так як Active Record - це ORM фреймворк, ви завжди працюєте з об'єктами. Щоб створити новий рядок таблиці, ви створюєте новий об'єкт класу і заповнюєте його змінні екземпляра значеннями. Варто зауважити, що все це Active Record робить за вас.

Перевірка даних перед приміщенням їх в таблицю - це перший крок в безпеці вашого проекту. Active Record надає перевірку Моделі. Дані можуть бути перевірені автоматично за допомогою готових методів.

Action View включає в себе логіку, необхідну для виведення даних Моделі. Роль Уявлення в RoR грає Action View. Функції ActionView:

Шаблони (Templates). Шаблони - це файли, що містять наповнювачі (placeholders), які буду замінені на контент. Шаблони можуть містити HTML-код і код Ruby, вбудований в HTML з використанням синтаксису вбудованого (embedded) Ruby (ERB).Помічники (helper, далі хелпер) форм і форматування. Хелпери форм дозволяють створювати такі елементи сторінок, як чекбокси, списки, використовуючи готові методи. У свою чергу хелпери форматування дозволяють форматувати дані за необхідне нам способом.

Макети (layouts) визначають, як контент буде розташований на сторінці. Динамічно створювана сторінка може містити вкладення з декількох сторінок, навіть без використання таблиць і фреймів, використовуючи API Макета.У веб-додатку Контролер регулює потік логіки додатка він знаходиться на кордоні програми, перехоплюючи всі запити, на основі яких він змінює якийсь об'єкт Моделі і викликає Вид, щоб відобразити оновлені дані. У RoR Action Controller є Контроллером, ось його основні функції для роботи програми:

Підтримка сесій. Сесія - період часу, проведений користувачем на сайті. Його можна відстежити за допомогою cookie або об'єкта сесії. Cookie - невеликий файл, він не може містити об'єкти.Бувають ситуації, коли необхідно викликати певний код, перед тим як виконувати логіку Контролера або після

нього, наприклад, аутентифікація користувачів, логування подій, надання персонального відповіді. Допомагають в таких випадках фільтри, що надаються Action Controller. Існують три основних фільтра: before, after і around.

Кешування - це процес, при якому найбільш запитуваний контент зберігається в кеші, щоб не запитувати його знову і знову.

1.8.2. Sinatra

Sinatra - це предметно-орієнтований каркас (DSL) для швидкого створення функціональних веб-додатків на Ruby з мінімумом зусиль. Sinatra є невеликим і гнучким додатоком, який, однак, не слідує типовому паттерну Model-View-Controller (модель-вид-контролер), який часто застосовується в веб-фреймворках, як наприклад в Ruby on Rails. Замість цього Sinatra фокусується на ідеї швидкого створення веб-додатків на Рубі з мінімальними зусиллями.Багато людей вбудовують Sinatra додатки в додатки на Rails. Це імітує дизайн Django фреймворка, де основний додаток складається з безлічі більш дрібних додатків, кожний з яких відповідає за певну частину в ньому (і часто можуть бути повторно використані іншими додатками).

У вас навіть може бути мікрододаток на Sinatra всередині Rails структури. Такиї підхід застосовується на Github для вирішення декількох завдань. Замість того, щоб підлаштовувати Rails під свою волю заради специфічної можливості, ви можете перенаправити запит на Sinatra або кінцеву точку Rack і виконати саме те що необхідно. Rails буде кращим варіантом в тому випадку, якщо вам просто необхідно розібратися із завданням. Для простих речей найкращим вибором буде Sinatra, а також тоді, коли ви хочете все контролювати і дотримуватися власних поглядів.Головним недоліком Sinatra, який не вирішує за вас проблему, є саме те, що Sinatra її не вирішує. Ви повинні самостійно з нею розібратися. Це може привести до того, що ви витратите невиправдано велика кількість часу, намагаючись її вирішити.

1.8.3. Rack

У 2007 році Крістіан Нойкірхен(Christian Neukirchen) написав бібліотеку Rack, яка надає єдиний API для всіх написаних на Ruby HTTP-серверів і веб-додатків.Майте на увазі, що Rack призначена не для високорівневою розробки,а для створення інструментів, бібліотек і каркасів. Крім того, вона розрахованана досвідчених програмістів, а користувачі каркаса або бібліотеки зазвичай небачать Rack API.

Якщо розібратися в призначенні і способах роботи з Rack, то в ваших руках виявиться потужний інструмент.За кілька років, які йшли з моменту його появи, написані на його основі програми обробили багато трильйонів запитів.Розглянемо деякі деталі роботи цієї бібліотеки. HTTP-запит, як мінімум, включає набір параметрів, організований у вигляді хеша. Повертаєма відповідь складається з трьох частин: код стану, набір заголовків і тіло. Припустимо, що у об'єкта є метод call, який повертає масив такого формату. Будь який класс написаний за описаними правилами, можна вважати Rack додатком. Ось простий приклад такого класу:

```
class MyRackApp
  def call ( env)
    [200, { 'Content-type' => 'text/plain' ) , [ "Welcome to Rack ! "]]
  end
end
```

Відзначимо, що тіло саме є масивом, а не просто рядком. пояснюється це угодою, за якою тіло повинно відповідати на виклик методу each. (Насправді, простий рядок теж згодився би, якби вона відповідала на виклик each,як було в старих версіях Ruby.) Ну і як запустити такий додаток? Способів кілька. До складу Rack входить, утиліта rackup. По суті справи, це виконавець додатків, що вміє працювати з файлами з розширенням .ru. Нижче наведено приклад такого файлу, який просто викликає додаток, передаючи методу run об'єкт в якості параметра.

```
app = MyRackApp.new
run app
```

За угодою, Rack-додаток конфігурується за допомогою написаного на Ruby файлу config.ru. Користуючись нескладним Rack API, ми можемо сильно сростити HTTP-сервер.

У наведеному нижче фрагменті та ж сама HTTP-відповідь надсилається засобами Rack. Зверніть увагу, наскільки лаконічніше стрілка, що позначає лямбда-вираз:

```
run -> (env) {
[ 200, {"Content-Type" => "text/html"}, ["", "Hello from Ruby!", ""]
] }
```

Будь-який Rack-додаток повинен відповідати на звернення до методу call, котрий викликається один раз для кожного запиту. Аргумент env, який передається методу call, містить інформацію про запит: дієслово HTTP, шлях, заголовки, IP-адреса клієнта і багато іншого. Оскільки будь-який proc-об'єкт вміє відповідати на виклик call, найпростішим Rack-додатком є лямбда, як в прикладі вище. Кожен Rack-додаток має повертати Rack-відповідь: масив з трьох елементів, що містить код його стану (число), заголовки відповіді (у вигляді хеша) і тіло (у вигляді об'єкта, що відповідає на виклики each; в нашому випадку це масив).

Важлива концепція, завдяки якій бібліотека Rack і є настільки корисною це ідея ПО проміжного рівня. Можна вважати, програма, розташована між браузером (або клієнтом) і сервером. вона опрацьовує запити, що відправляються клієнтами сервера, і відповіді сервера клієнтам.

При цьому компоненти проміжного рівня можна більш-менш довільно але “зчіплювати”. Кожен наступний компонент якимось чином перетворює потік даних. Можна розглядати це як аналог конвеєра в UNIX.Існують угоди і прийоми написання компонентів проміжного рівня, але тут ми їх розглядати не будемо. Відзначимо лише, що є безліч готових компонентів - як що входять до складу Rack. так і написаних сторонніми розробниками. Вони реалізують,

наприклад, HTTP-кешування, фільтрацію спаму, перевірку працездатності відправкою періодичного сигналу-пульсу, аналітику Google і багато іншого.

Якщо ви збираєтеся серйозно використовувати Rack, то повинні познайомитися з класом Rack::Builder; це предметно-орієнтована мова для складання Rack додатків з окремих шматочків. Утиліта rackup перетворює ru-файли в об'єкт Builder, але, якщо потрібна велика гнучкість і додаткові можливості, то ви можете зробити це і самостійно.Для запуску Rack-додатки можна використовувати будь-який з численних сумісних з Rack HTTP-серверів, написаних на Ruby. Цим ви звільните себе від необхідності писати і підтримувати TCP-сервер, коректно розбивати HTTP-запити і форматувати HTTP-відповіді.

Якщо жоден інший сервер не встановлено, то Rack за замовчуванням візьме сервер Webrick, що входить до складу дистрибутива Ruby, але для виробничих програм він не годиться. Самий популярний написаний на Ruby веб-сервер - Unicorn, він оптимізований під однопоточні додатки для інтерпретатора MRI. додаткові відомості про Unicorn можна знайти в документації на сайті unicorn.bogomips.org хоча багато розробників віддають перевагу веб-серверу Puma, який підтримує багатопотоковість, а також інтерпретатори MRI, JRuby і Rubinius. додаткову інформацію про нього див. на сайті puma.io. Зверніть також увагу на повідомлення, що друкуються на консолі сервером Puma і бібліотекою Rack. Сервер виводить один рядок для кожного запиту, яка включає IP-адресу відправника, дату і час, дієслово HTTP, версію протоколу HTTP, повернутий код стану і час, витрачений на обробку запиту.

1.8.4. Middleman

Генератор статичних сайтів Middleman організовує оточення, дуже схоже на наявне в додатку Rails, тільки результатом його роботи є набір файлів, які може віддавати будь HTTP-сервер. Щоб скористатися Middleman, спочатку встановіть gem-пакет командою gem install middleman. В склад пакету входить утиліта middleman, що надає кілька команд для керування створенням статичних сайтів.

Для отримання повного списку команд введіть middleman help.Установів Middleman, створимо новий проект командою middleman init hello_world. В результаті буде створений каталог hello world, а в ньому файли, що становить сайт (майже порожній).

Щоб подивитися, як -виглядіт сайт, перейдіть в цей каталог командою cd, виконайте команду bundle start middleman server, яка запустить сервер попереднього перегляду, а потім перейдіть в браузері за адресою http://localhost:4567. Як і сервер Rails, сервер Middleman дозволяє швидко оцінювати результати внесених в сайт змін на етапі розробки.

Закінчивши роботу над сайтом, ми можемо зупинити сервер Middleman і виконати команду middleman build. Вона створить остаточний статичний сайт і помістіть всі файли в підкаталог puИіс всередині каталогу проекту Middleman. згенеруванний сайт можна розгорнути на робочому сервері, просто передавши файли з каталогу public на потрібний HTTP-сервер. Оскільки ми хочемо, щоб наш сайт показував вітання, змінимо файл source/index.html. erb, помістивши в нього наступний текст:

title: Hello from Middleman!

<Hl> Hello from Middleman! </ Hl>

Збережіть файл і перезавантажте сторінку http://localhost: 4567, щоб подивитися, що вийшло. Оскільки Middleman створює статичні сайти, в URI, ми можемо вказувати тільки шляхи, відповідні фізичних файлів. щоб додати привітання, розміщене за адресою /world, ми повинні створити файл.Спочатку створіть новий каталог командою mkdir source /, юrld. Потім створіть в ньому

файл source/world/index.html.erb і помістіть в нього повідомлення: <Hl> Hello, World! </ Hl> Тепер, перейшовши за адресою http://localhost:4567/world, ви побачите це привітання.

Напевно, у вас виникло питання, навіщо ми створили каталог world і помістили в нього шаблон index.html.erb. Причина - в бажанні мати гарні URL-адреси.Сервери, які роздають статичні файли, шукають файл на ім'я, що включає і розширення. Якщо в запиті вказано шлях, який не містить розширення, наприклад /world, то сервер буде переглядати відповідний каталог.

Якщо він знайде в ньому індексний файл, то автоматично поверне його. Саме тому, помістивши індексний файл в каталог world, ми змогли побачити привітання, перейшовши за адресою /world.

Якби ми просто створили файл world.html.erb, то змогли б побачити привітання, лише вказавши шлях /world.html.Стаиичні сайти, згенеровані Middleman, надають, в основному, ті ж утиліти і допоміжні засоби, що і додатки Rails, в тому числі допоміжні функції в шаблонах, повний конвеєр активів, включаючи інтеграцію з Sass, Compass і CoffeeScript, і багато іншого.

Повний перелік функцій див. в документації по Middleman на сайті middlemanapp.com

1.9. Загальні web-технології

1.9.1. HTML/HAML

HTML (від англ. Hyper Text Markup Language) - стандартна мова розмітки документів у Всесвітній павутині. Більшість веб-сторінок містять опис розмітки на мові HTML (або XHTML). Мова HTML інтерпретується браузерами і відображається у вигляді документа в зручній для людини формі.Мова HTML є додатком SGML (стандартної узагальненої мови розмітки) і відповідає міжнародному стандарту ISO 8879. У свою чергу XHTML є більш суворим варіантом HTML, він слідує всім обмеженням XML і, фактично, XHTML можна сприймати як додаток мови XML до області розмітки гіпертексту.

У всесвітній павутині HTML-сторінки, як правило, передаються браузерам від сервера по протоколах HTTP або HTTPS, у вигляді простого тексту або з використанням шифрування.

Текстові документи, що містять розмітку на мові HTML (такі документи зазвичай мають розширення .html), обробляються спеціальними додатками, які відображають документ в його форматованому вигляді. Такі додатки, звані «браузерами» або «інтернет-оглядачами», зазвичай надають користувачеві зручний інтерфейс запиту веб-сторінок, їх перегляду (і виведення на інші зовнішні пристрої) і, при необхідності, відправки введених користувачем даних на сервер.

Haml (HTML abstraction markup language) - мова розмітки для спрощеної генерації HTML. HAML компілюється в HTML.

1.9.2. CSS/SCSS

CSS (англ. Cascading Style Sheets - каскадні таблиці стилів) - формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки переважно використовується як засіб опису, оформлення зовнішнього вигляду веб-сторінок.

CSS використовується творцями веб-сторінок для завдання кольорів, шрифтів, розташування окремих блоків і інших аспектів представлення зовнішнього вигляду цих веб-сторінок. Основною метою розробки CSS було розділення опису логічної структури веб-сторінки (яке проводиться за допомогою HTML або інших мов розмітки) від опису зовнішнього вигляду цієї веб-сторінки (яке тепер проводиться за допомогою формального мови CSS).

Такий поділ може збільшити доступність документа, надати велику гнучкість і можливість управління його поданням, а також зменшити складність і повторюваність в структурному вмісті. Крім того, CSS дозволяє представляти один і той же документ в різних стилях або методах виведення, таких як екранне уявлення, друковане подання, читання голосом (спеціальним голосовим браузером або програмою читання з екрану), або при виведенні пристроями, що використовують шрифт Брайля.Як відомо, HTML-документи будуються на підставі ієрархії елементів, яка може бути наочно представлена в деревовидної формі. Елементи HTML один для одного можуть бути батьківськими, дочірніми, елементами-предками, елементами-нащадками, сестринськими.

Елемент є батьком іншого елемента, якщо в ієрархічній структурі документа він знаходиться відразу, безпосередньо над цим елементом. Елемент є предком іншого елемента, якщо в ієрархічній структурі документа він знаходиться десь вище цього елемента.

Нехай, наприклад, в документі присутні два абзаци p, що включають в себе шрифт з напівжирним шрифтом b. Тоді елементи b будуть дочірніми елементами своїх батьківських елементів p, і нащадками своїх предків body.

У свою чергу, для елементів p елемент body буде тільки батьком. І крім того, ці два елементи p будуть сестринськими елементами, як мають одного і того ж батька - body. В CSS можуть задаватися за допомогою селекторів не лише поодинокі елементи, але і елементи, які є нащадками, дочірніми або сестринськими елементами інших елементів.

Sass (Syntactically Awesome Stylesheets) - модуль, що входить в Haml. Sass - це метамова на основі CSS, призначений для збільшення рівня абстракції CSS коду та спрощення файлів каскадних таблиць стилів.

1.9.3. JavaScript/CoffeeScript

JavaScript - прототипна-орієнтована скриптова мова програмування. Є діалектом мови ECMAScript. JavaScript зазвичай використовується як вбудований мова для програмного доступу до об'єктів додатків.

Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок. Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу.На JavaScript вплинули багато мов, при розробці була мета зробити мову схожим на Java, але при цьому легким для використання не програмістів. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє його від ряду мов програмування, використовуваних в Інтернет.

CoffeeScript - мова програмування, що транслюється в JavaScript. CoffeeScript додає синтаксичний цукор в дусі Ruby, Python, Haskell і Erlang для того, щоб поліпшити читаність коду і зменшити його розмір. CoffeeScript дозволяє писати більш компактний код в порівнянні з JavaScript. JavaScript-код, отриманий трансляцією з CoffeeScript, повністю проходить перевірку JavaScript Lint.

1.9.4. jQuery

jQuery – бібліотека JavaScript, що фокусується на взаємодії JavaScript і HTML. Бібліотека jQuery допомагає легко отримувати доступ до будь-якого елементу DOM, звертатися до атрибутів і вмісту елементів DOM, маніпулювати ними. jQuery надає зручний API для роботи з AJAX.

Точно так же, як CSS відокремлює візуалізацію від структури HTML, Jquery відокремлює поведінку від структури HTML. Наприклад, замість прямої вказівки на обробник події натискання кнопки управління передається Jquery, яка ідентифікує кнопки і потім перетворює його в обробник події кліка. Такий поділ поведінки і структури також називається принципом ненав’язливого JavaScript.

Бібліотека jQuery містить функціональність, корисну для максимально широкого кола завдань. Проте, розробниками бібліотеки не ставилося завдання суміщення в jQuery функцій, які підійшли б усюди, оскільки це призвело б до великого коду, велика частина якого не затребувана. Тому була реалізована архітектура компактного універсального ядра бібліотеки і плагінів. Це дозволяє зібрати для ресурсу саме ту JavaScript-функціональність, яка на ньому була б затребувана.

1.9.5. Git

Git - розподілена система керування версіями файлів.Система спроектована як набір програм, спеціально розроблених з урахуванням їх використання в скриптах. Це дозволяє зручно створювати спеціалізовані системи контролю версій на базі Git або призначені для користувача інтерфейси. Наприклад, Cogito є саме таким прикладом оболонки до репозиторіїв Git, StGit використовує Git для управління колекцією виправлень.Git підтримує швидке поділ і злиття версій, включає інструменти для візуалізації та навігації по нелінійної історії розробки. Як і Darcs, BitKeeper, Mercurial, Bazaar і Monotone, Git надає кожному розробнику локальну копію всієї історії розробки, зміни копіюються з одного сховища в інший.Віддалений

доступ до репозиторіїв Git забезпечується git-daemon, SSH- або HTTP-сервером. TCP-сервіс git-daemon входить в дистрибутив Git і є поряд з SSH найбільш поширеним і надійним методом доступу.

Метод доступу по HTTP, незважаючи на ряд обмежень, дуже популярний в контрольованих мережах, тому що дозволяє використовувати існуючі конфігурації мережевих фільтрів.Ядро Git є набором утиліт командного рядка з параметрами. Всі налаштування зберігаються в текстових файлах конфігурації. Така реалізація робить Git легко портуємість на будь-яку платформу і дає можливість легко інтегрувати Git в інші системи (зокрема, створювати графічні git-клієнти з будь-яким бажаним інтерфейсом).

Репозиторій Git є каталог файлової системи, в якому знаходяться файли конфігурації сховища, файли журналів, що зберігають операції, що виконуються над репозиторієм, індекс, що описує розташування файлів і сховище, що містить власне файли.Структура сховища файлів не відображає реальну структуру зберігається в репозиторії файлового дерева, вона орієнтована на підвищення швидкості виконання операцій з репозиторієм. Коли ядро обробляє команду зміни (неважливо, при локальних змінах або при отриманні патча від іншого вузла), воно створює в сховище нові файли, що відповідають новим станам змінених файлів.Істотно, що ніякі операції не змінюють вмісту вже існуючих в сховище файлів.За замовчуванням репозиторій зберігається в підкаталозі з назвою «.git» в кореневому каталозі робочої копії дерева файлів, що зберігається в репозиторії.Будь-яке файлове дерево в системі можна перетворити в репозиторій git, віддавши команду створення сховища з кореневого каталогу цього дерева.

РОЗДІЛ 2. РОЗРОБКА І ДОСЛІДЖЕННЯ МЕТОДИКИ ДИНАМІЧНОЇ ГЕНЕРАЦІЇ ВЕБ-САЙТІВ

2.1. Визначення функціоналу динамічного генератора веб-сайтів

Під час дослідження предметної області яке я виклав у першому розділі даної магістерської дипломної роботи я дійшов висновку що для успішної підтримки функціонування багаторівневої системи веб-сайтів з використанням дешевих серверних послуг та підвищеним рівнем безпеки необхідні нові інструменти. Приведені технології мають свої сильні та слабкі сторони, та беручи до уваги всі засоби та технології дозволю собі зупинитись на думці що статичні сторінка мають безумовну перевагу у швидкості та безпеці. При цьому можна розширити розуміння статичної сторінки перейшовши від простої HTML сторінки яка віддається до браузеру користувача безпосередньо такою як вона є до більш складної але на мою думку досі статичної структури коли сторінка все ж генерується перед поверненням до користувача та її зміст та стан може бути заданий лише за допомогою зовнішнього контролю і не може бути змінений навіть якщо у зловмисника є доступ до файлів проекту. Така псевдо статичність при якій все ж відбувається накладення схеми даних на певний шаблон, ставить собою на меті убезпечення даних нашого клієнту від несанкціонованого змінення.

При цьому окрім змінення важливим моментом являється такий стан кінцевого клієнта, до якого звертається користувач, що жодні данні не можуть зберігатись на ньому безпосередньо, зберігаються в зашифрованому вигляді або у вигляді надлегких JSON файлів відповідно до мети мережі та рівня необхідної безпеки заданого при її розгортанні. Часто можливими приводами для блокування певних сайтів є їх вміст, авторитарні режими по всьому світу блокують доступ до інформації за допомогою судових рішень, що на мою думку є неприйнятним у 21 сторіччі, при цьому сервери в цілому або окремі данні на них вилучаються, що може призвести до їх втрати. Раціональним рішенням на мою думку є часткова або повна відмова від збереження даних на

ресурсах які знаходяться під загрозою закриття або видалення. Окрім втрати інформації через фізичний або віртуальний доступ до серверу та блокування ресурсу скажімо за рішенням суду атака на інформаційну мережу може здійснювати за допомогою так званих DDOS атак(англ. Distributed Denial-of-service attack) так званих атак на відмову в обслуговуванні. Скажу окремо що під такої атакою розуміється напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена. Одним із найпоширеніших методів нападу є насичення атакованого комп'ютера або мережевого устаткування великою кількістю зовнішніх запитів (часто безглуздих або неправильно сформульованих) таким чином атаковане устаткування не може відповісти користувачам, або відповідає настільки повільно, що стає фактично недоступним. Тож при критичній необхідності при розповсюджені певної інформації ризики мультиваріативні і якщо гостро стоїть необхідність у миттєвій віддачи новин чи даних мережа має мати у собі “сплячі” елементи які неактивовані в даний момент і можуть бути введені в експлуатацію при необхідності. Так вимкнені клієнти не можуть бути просто заблоковані за допомогою адміністративного чи іншого тиску через то що як було відмічено вище не зберігають жодної інформації та не мають жодного аналогу стартової сторінки. Весь акцент системи лежить на розподіленості, маскуванні та на властивості працювати з видаленими даними.

Розглянувши всі ввідні виникає питання чим буде відрізнятись така мережа сайтів або окремий екземпляр від банального скрипту RSS стрічки налаштованого на певне джерело який би просто робив окремий внутрішній запит до нього при запиті з браузеру. Відмінність в тому що і сама конфігурація і програмні властивості клієнту можуть бути отримані як дані, навіть самі джерела контенту можуть бути отримані з конфігурації, клієнт може отримати свої налаштування не лише від певного серверу адміністрування але й може сам отримати у вигляді даних модулі контролю для керування іншими

клієнтами або модулі та данні для того щоб клієнт сам став джерелом даних для інших членів мережі(Рис.2.1).

Рис.2.1 Варіанти використання кожного окремого клієнту

Варто зазначити що я відокремлюю роль джерела даних та серверу управління одне від одного, звичайно у класичній REST системі (Рис.2.2) це може бути одним і тим же ресурсом і така схема дійсно може працювати і тут, але для більш гнучкої, роботи конфігурації на клієнт можуть бути завантажені з будь якого засобу управління будь то мобільний додаток, десктопний клієнт, встановлена в веб панель адміністрування чи простий запит з командної строки.

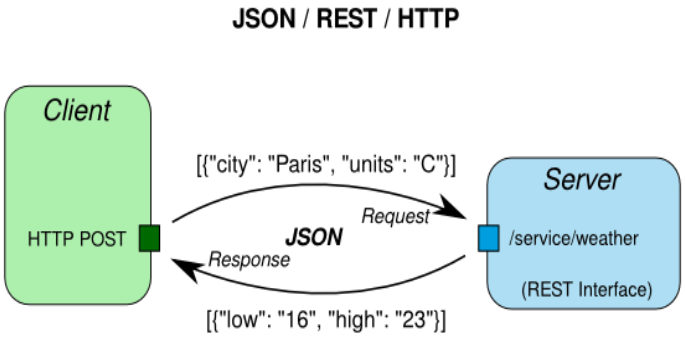

Рис.2.2 Загальна схема REST роботи клієнта та сервера

Маючи певний набір властивостей можливо врешті решт сформулювати визначення. Під динамічним генератором веб-сайтів я розумію клієнт-серверну архітектуру в якій клієнт конфігурації який може виглядати як будь яка система здатна надсилати POST запити виконує роль CMS яка змінює стани своїх клієнтів, видаляє чи змінює вміст або функціональне призначення. На клієнтському домені встановлена, базована на метапрограмуванні система яка може за командою від сервера змінювати не лише вміст але й власну структуру. Задля спрощення системи та збільшення можливостей можливо щоб вміст сторінок, конфігурацій безпосередньо у серверному кешу або за допомогою гнучкої системи у форматі JSON файлів для розгортання на більш простих серверах.

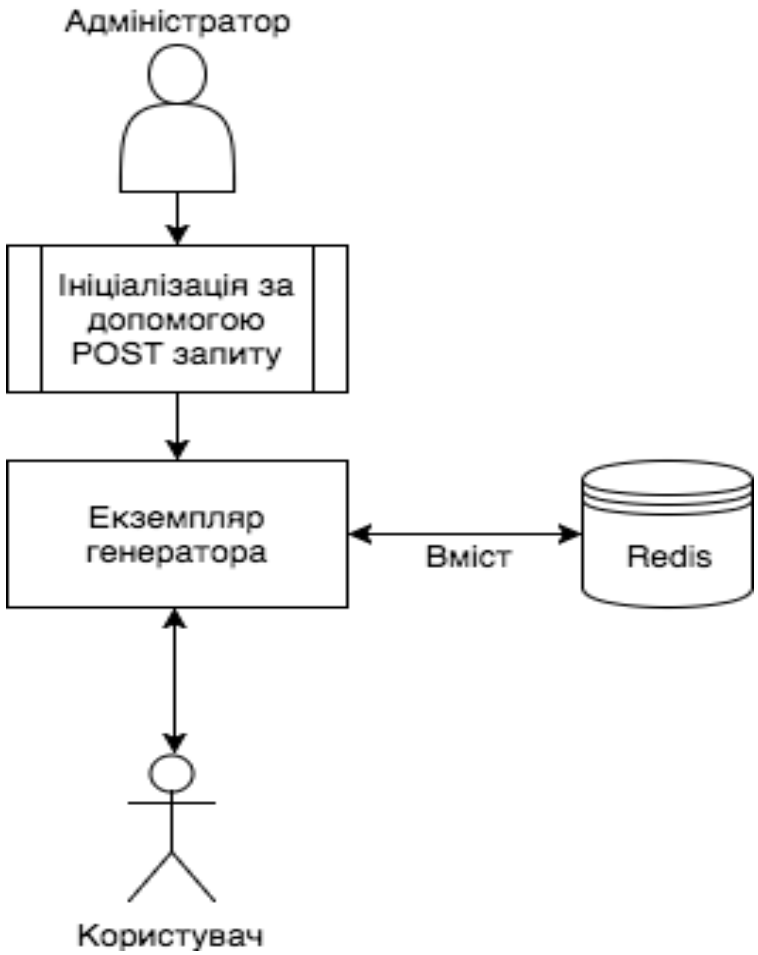

Рис.2.3 Спрощена схема взаємодії з окремим екземпляром динамічного генератора

Виконуючи команди зміни контенту по регіонам чи сегментам адміністратор динамічного генератора впливає на цілу мережу ресурсів. На прикладі лендінгів ми можемо використовувати перелінковку та зміну SEO параметрів щоб швидко реагувати на зміни у кон'юнктурі ринку. В такому разі наші клієнти не будуть в повну міру статичними в плані програмного забезпечення яке буде на них встановлено, та динамічне керування буде більш гнучким ніж проста зміна вмісту десятків веб-сайтів через систему автоматизації роботи з FTP, адже в такій ситуації все ж комусь доведеться у ручному чи напівавтоматичному режимі вносити правки у всі ці сторінки.

Розглянемо переваги даної моделі перед звичним мультисайтингом. Динамічний генератор дозволить задіяти не лише зміну контенту на одному з наших клієнтів, як це пропонують деякі системи для цього пристосовані, але я саму сутність нашого віддаленого клієнту, в тому числі і його функцію, створити на ньому окремі ендпоінти чи окремі нові веб-сервіси для провадження наступних запитів. За допомогою метапрограмування можна розширювати функціональні можливості наших керованих доменів.

Вище була розглянута push-схема циркуляції даних яка потребує внесення змін у клієнтські домени за спеціальною командою. На противагу їй pull-схема влаштовує роботу таким чином коли налаштовані клієнти самі опитують сервери, на наявність змін у неулаштуваннях чи контенту, за такою схемою ми можемо позбавити клієнти POST інтерфейсів які можуть бути не безпечними у певних ситуаціях, і організувати з них мережу “слухачів” які б змінювали свій стан періодично за внутрішнім розкладом.

В сучасному світі інформація набуває особливої цінності не лише завдяки бізнес процесам, на тлі розширення глобального громадянського суспільства, захист та поширення даних мають дуже високу актуальність. Якщо ви хочете зберегти доступ до певного контенту поширюючи його по сотням доменів чи просто прагнете відокремити своїх потенціальних споживачів в новій рекламній кампанії, методика динамічної генерації веб-сайтів дозволить зробити цю роботу зручною та ефективною.

2.2. Зберігання даних

На мою думку головним питанням для динамічного генератора веб-сайтів є зберігання даних, якого б значення вони не мали чи це буде простий текст для розповсюдження чи програмний код для розгортання сервісу певного типу або навіть зображення конвертовані в base64 код. Ситуація з використанням файлового сховища на кшталт JSON чи TXT файлів робить веб-сайт повільним, та складним для адміністрування і безпеки процес перезапису даних в цих файлах. Одразу відмічу що реляційні бази даних мною не розглядались через те що це потребувало би створення певної схеми бази даних при ініціалізації та маніпулюванні цією схемою при подальшій необхідності в зміні напряму роботи веб-сайту. І хоча реляційні бази більш поширені серед продуктів компаній які пропонують дешевий хостинг, для динамічного генератора вибір залишається між цими двома типами. Звичайно завжди можливо залишити конфігурацію яка б дозволила працювати в режимі ретранслятора виконуючи запити до видалених ресурсів після кожного запиту користувача на власні ендпоінти. Тому мені здається найбільш кращим варіантом використання сховища яке зберігає дані в оперативній пам’яті. Серед таких сховищ варто виділити два Memcached та Redis.

Memcached — комп'ютерна програма, сервіс кешування даних в оперативній пам'яті на основі парадигми розподіленої хеш-таблиці. За допомогою клієнтської бібліотеки (для Perl, PHP, Python, Java та ін.) дозволяє кешувати дані в ОЗП одного або декількох серверів. Розподіл даних реалізується по значенню хеш ключа. Використовуючи ключ даних, клієнтська бібліотека визначає його хеш і використовує його для вибору відповідного сервера. Ситуація збою сервера трактується як промах кеша. Це дозволяє, зокрема, проводити гарячу заміну серверів. В API memcached є тільки базові функції: вибір сервера, установка з'єднання, додання, видалення, оновлення і отримання об'єкта. Для кожного об'єкта встановлюється час актуальності, починаючи з 1 секунди до нескінченності. При переповненні пам'яті застарілі об'єкти кеша автоматично знищуються.

Redis(Рис.2.4) - розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання за бажанням користувача. Це програмне забезпечення з відкритим сирцевим кодом написане на ANSI C. Розробка Redis фінансується VMware. Сирцеві тексти проекту поширюються в рамках ліцензії BSD. Redis надає схожі на Memcached функції для зберігання даних в форматі ключ/значення, розширені підтримкою структурованих даних, таких як списки, хеші і множини.

Рис.2.4 Розподілене сховище Redis

На відміну від Memcached, Redis забезпечує постійне зберігання даних на диску і гарантує збереження БД у разі аварійного завершення роботи. Клієнтські бібліотеки доступні для більшості популярних мов, включаючи Perl, Python, PHP, Java, Ruby і Tcl. Є підтримка транзакцій, що дозволяють виконати за один крок групу команд, гарантуючи несуперечність і послідовність (команди від інших запитів не можуть вклинитися) виконання заданого набору команд, а в разі проблем дозволяючи відкотити зміни. Всі дані у повному обсязі кешуються в оперативній пам'яті. Зберігання всіх даних в оперативній пам'яті дозволяє досягнути значної продуктивності: при тестуванні Redis на сервері з CPU Xeon X3320 2.5 ГГц вдалося забезпечити 110000 операцій запису і 81000 операцій читання за секунду. Для управління даними підтримуються такі команди, як інкремент/декремент, стандартні операції над списками і множинами (об'єднання, перетин), перейменування ключів, множинні вибірки та функції сортування. Підтримується два режими зберігання: періодична синхронізація даних на диск і ведення на диску логу змін. У другому випадку

гарантується повне збереження всіх змін. Можлива організація master-slave реплікації даних на кілька серверів, здійснювана в неблокуючому режимі

На зовнішньому рівні абстракції, модель даних в Redis це асоціативний масив в якому ключі відображаються в значення. Основною відмінністю між Redis та іншими базами такого типу в тому, що значення словника не обмежені рядковими типами. На додачу до рядків підтримуються наступні абстрактні типи даних(Рис.2.5):

- Списки рядків
- Множини рядків (невпорядкований набір неповторюваних елементів)
- Впорядковані множини рядків (набори неповторюваних елементів впорядкованих за пов'язаним значенням з плаваючою комою)
- Хеші де ключі і значення є рядками
- Redis підтримує високорівневі атомні операції на стороні сервера, такі як перетин, об'єднання та різниця між множинами та списками.

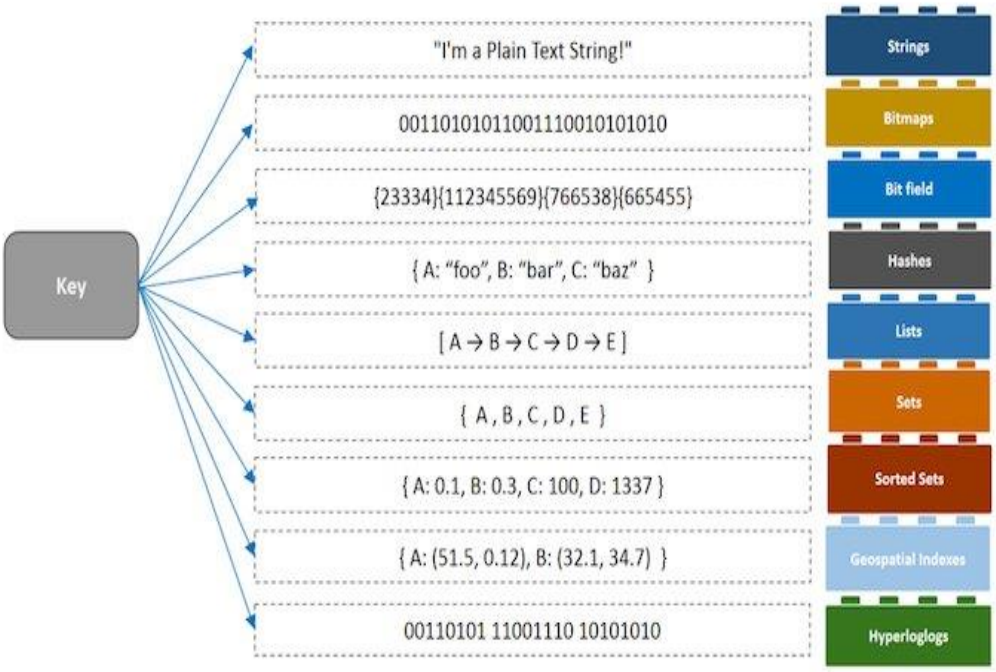

Рис.2.5 Типи даних в Redis

Redis зазвичай тримає всі дані в оперативній пам'яті. До версії 2.4 існувала можливість налаштувати роботу з віртуальною пам'яттю але зараз це

не рекомендовано. Персистентність досягається двома шляхами, перший з яких називається знимкування (англ. snapshotting), і є напівперсистентним режимом довговічності, в якому дані час від часу асинхронно передаються з пам'яті на диск. З версії 1.1 безпечнішою альтернативою є файл який можна лише доповнювати (лог транзакцій), і в який доповнюється всіма операціями що модифікують дані в пам'яті. Redis має можливість переписувати файл з нуля в фоновому режимі для того щоб уникнути нескінченного росту його довжини.

Користуючись власним досвідом та рекомендаціями в мережі я вважаю що краще упинитись на сховищі Redis та використовувати його широкі можливості у сфері типів даних для всіх необхідних елементів які необхідно зберігати для нормального функціонування сайту. При цьому особливо цікавим є те що від версії 2.6 у Redis був вбудований інтерпретатор Lua і підтримуються скрипти на мові Lua(Рис.2.6), що працюють на стороні сервера.

Рис.2.6 Мова програмування Lua

Коротко про Lua ([лу́а], порт. місяць) — швидка і компактна скриптова мова програмування, розроблена підрозділом Tecgraf Католицького університету Ріо-де-Жанейро (Computer Graphics Technology Group of Pontifical

Catholic University of Rio de Janeiro in Brazil). Є вільно-поширюваною, з відкритим сирцевим кодом на мові Сі. За можливостями, ідеологією і реалізацією, мова найближча до JavaScript, проте Lua відрізняється могутнішими і набагато гнучкішими конструкціями, спроектованими з метою «не плодити сутності понад необхідне». Хоча Lua не містить поняття класу і об'єкта в явному вигляді, механізми об'єктно-орієнтованого програмування з підтримкою прототипів (включаючи множинне успадкування) легко реалізуються з використанням метатаблиць, які також дозволяють перевантаження операцій, тощо. Реалізована модель ООП (як і в JavaScript) — прототипна.

Lua отримала велике поширення в ролі вбудованої в інші проекти мови сценаріїв (наприклад, для визначення конфігурації або для написання розширень). Lua комбінує простий процедурний синтаксис з потужними можливостями опису даних через використання асоціативних масивів і розширюваної семантики мови. У Lua використовується динамічна типізація, мовні конструкції перетворюються на байт-код, який виконується поверх регістрової віртуальної машини з автоматичним збирачем сміття. Сам інтерпретатор оформлений у вигляді бібліотеки, легко інтегрованої в проекти на мовах Сі та Сі++. Код інтерпретатора Lua написаний на мові Сі і розповсюджується під ліцензією MIT.

За допомогою Lua ми отримаємо гнучку систему скриптингу для написання запитів для роботи з даними які зберігаємо у Redis, якщо розширити цю тему то виконання то багато хто знає про можливість зберігати процедури в sql базах даних, про це написано чимало пухких посібників і статей. Але так як Redis не реляційна БД, то і принципи опису процедур досить сильно відрізняються. Збережені процедури в Redis - практично повноцінні Lua скрипти.

2.3. Метапрограмування динамічних функцій

Метапрограмування в скриптових мовах - це стиль написання програм, при якому з користю використовуються можливості зміни простору імен в runtime. Під простором імен мається на увазі класи, методи і змінні (глобальні, локальні, змінний примірника і змінні класу). Зміна означає створення, зміна та видалення класів, методів і змінних. Треба сказати, що в більшості скриптових мов простір імен конструюється не інакше як в режимі runtime. Використання даної техніки в динамічному генераторі сайтів дозволить додавати нові модулі та класи з новими функціями прямо у процес виконання, даючи змогу екземпляру змінювати поведінку екземпляра прямо під час виконання. Слід детальніше розглянути роботу методу eval у Ruby та інших вбудованих механізмів.

Глобальна функція eval компілює і виконує рядок, що містить код на Ruby. Це дуже потужний (і те місце з тим небезпечний) механізм, оскільки дозволяє будувати виконуваний код під час роботи програми, наприклад в наступному фрагменті зчитуються рядки виду “ім'я = вираз”, потім кожен вираз обчислюється, а результат зберігається в хеші, індексованим ім'ям змінної.

```
parameters = (}
ARGF.each do |line|
  name, expr = line.split(/\s*=\s*/, 2)
  parameters[name] = eval expr
end
```

Нехай на вхід подаються наступні рядки:

```
a = 1
Ь = 2 + 3
c= 'date'
```

Тоді в результаті ми отримаємо такий хеш: {"a" => :, "b" => 5, "c" => "Sat Jul 21 00: 51: 48 "). На цьому прикладі демонструється також небезпека обчислення

за допомогою eval рядків, вміст яких ви не контролюєте; зловмисний користувач може підсунути рядок d = "rm" стерти все зроблене вами за день.

Блоки дозволяють нам групувати код і передавати його у вигляді аргументу для методу. Їх можна описувати за допомогою конструкції do end або фігурних дужок. Обидва варіанти тотожні.

Метод send дозволяє нам викликати методи об'єкту (навіть приватні), передаючи йому ім'я методу у вигляді символу. Це корисно для виклику методів, які зазвичай викликаються всередині класу або для інтерполяції змінних для динамічних викликів методу.

В Ruby define_method дає нам можливість створювати методи не використовуючи звичайну процедуру при описі класу. Він приймає в якості аргументів рядок, яка буде ім'ям методу і блок, який буде виконуватися при виклику методу. Не менше важливим за нього для нас є instance_eval це річ, необхідна при створенні DSL майже так само, як і блоки. Він приймає блок і виконує його в контексті об'єкта-приймача. наприклад:

```
class MyClass
  def say_hello
    puts 'Hello!'
  end
end
MyClass.new.instance_eval {say_hello} # => 'Hello!'
```

Для розуміння корисно самостійно познайомиться з поняттям блоку, методом Proc#call, конструкцією lambda, а також з поняттями змінної примірника класу (instance variables - змінні, чиї імена починаються на собаку) і змінної класу (class variables - змінні, чиї імена починаються на дві собаки):

Proc - клас для блоків, які можна про себе називати неіменованого (анонімними) методами, які можна створювати прямо в виразах;

Вираз b.call (* args) виконує блок b, і повертає результат виконання; замість call можна використовувати квадратні дужки.

lambda {| a, ... | ...} - створює блок, наприклад b = lambda {| x, y, z | x + y + z} створить блок, який складає три числа, зокрема вираження b [1,2,3] поверне 6;

блоки створюються не тільки за допомогою lambda, вони також конструюються автоматично при виклику методу з подальшою конструкцією {...} або do ... end; наприклад ary.inject {| a, b | a * b} передасть всередину методу inject блок, що виконує множення двох чисел;

instance-змінні живуть в об'єктах і вважаються ініціалізувати значенням nil за замовчуванням;

class-змінні живуть в класах і вважаються за замовчуванням неініціалізованих; при їх використанні в вираженні без попередньої ініціалізації виникає Exception "uninitialized class variable .. in ...";

Виклик method_missing це та магія, завдяки якій працюють методи find_by_ * в Rails. Будь-який виклик невизначеного методу потрапляє в method_missing, який приймає на вхід ім'я викликаного методу і все передані йому аргументи. Це ще одна чудова річ тому що вона дозволяє створювати методи динамічно, коли ми не знаємо, що може бути реально викликано. Це дає нам можливість створити дуже гнучкий синтаксис.

Важливим для обробки отриманого як текст коду я розуміння контексту викликів, контекст це з певної точки зору простір імен, з якої щось видно, щось невидно, а щось видно по-своєму. Наприклад, з тіла методу видно instance-змінні того об'єкта, для якого цей метод видно, а self дорівнює цьому об'єкту. Instance-змінні інших об'єктів не видно. Особливу вираз self корисно розглядати як певний метод, який в кожному контексті може бути по-своєму визначений. Привід для зміни контексту - конструкції def і class. Саме вони зазвичай призводять до зміни видимості instance-змінних, class-змінних і зміні значення виразу self.Звичайний блок також є новим контекстом, нехай і включає в себе контекст, в якому був створений. У блоці можуть бути свої локальні змінні (також як в Сі) і аргументи (які слід інтерпретувати як особливі локальні змінні).Власне поняття контекст має своє цілком конкретне

відображення в Ruby - це об'єкт класу Binding. У кожного блоку є binding, і цей binding можна передавати як другий аргумент методу eval: «виконай даний код в такому то контексті». Гнучкість поведінки Ruby під час виконання не обмежується розглянутими вище динамічними можливостями. Серед іншого можна дізнатися ім'я функції яка викликає код і автоматично перерахувати всі визначені користувачем елементи (об'єкти, класи і функції). Наявність таких гнучких засобів дослідження і зміни елементів програми під час виконання спрощує вирішення багатьох завдань. Виконуюча система Ruby повинна відстежувати всі відомі об'єкти (хоча б для того, щоб прибрати сміття, коли на об'єкт більше немає посилань). Інформацію про них можна отримати за допомогою методу ObjectSpace.each_object. Якщо задати клас або модуль в якості параметра eacr1_object. то будуть повернуті лише об'єкти зазначеного типу. Програма на Ruby може стежити за власним виконанням. У цій можливості є багато застосувань: велика їх частина пов'язана з профілюванням. У деяких випадках бібліотеки взагалі не потрібні, тому що завдання можна вирішити на Ruby, скориставшись інтроспекцією. Такий підхід різко знижує продуктивність (і тому в режимі експлуатації не рекомендується), але, тим не меньш клас TracePoint дозволяє викликати написаний нами код при виникненні події яка нас зацікавила.

Як видно з цього розділу інструментарій Ruby дозволяє “на ходу” швидко змінювати поведінку програми та додавати нові класи та методи до вже ініціалізованих, це створює можливість отримувати сирцевий код у вигляді строки прямо з POST запиту та інтегрувати його у виконання екземпляру. При цьому звичайно можна наразитись на небезпеку отримання коду від зловмисників, та для цього необхідно використовувати засоби авторизації та захисту екземпляру динамічного генератора веб-сайтів.

2.4. Автентифікація для конфігурування

Задля вирішення питання автентифікації найкраще підходить схема з використанням ключа доступу, при ініціалізації екземпляру динамічного генератора ключ має бути записаний у файл конфігурації і за його допомогою має відбуватись підтвердження права власності на даний ресурс. Цей спосіб найчастіше використовується для автентифікації пристроїв, сервісів або інших додатків при зверненні до веб-сервісів.

Тут в якості секрету застосовуються ключі доступу (access key, API key) - довгі унікальні рядки, що містять довільний набір символів, по суті замінюють собою комбінацію username/password. У більшості випадків, сервер генерує ключі доступу за запитом користувачів, які далі зберігають ці ключі в клієнтських додатках.

При створенні ключа також можливо обмежити термін дії і рівень доступу, який отримає клієнтську програму при автентифікації за допомогою цього ключа. Хороший приклад застосування автентифікації по ключу - хмара Amazon Web Services. Припустимо, у користувача є веб-додаток, що дозволяє завантажувати і переглядати фотографії, і він хоче використовувати сервіс Amazon S3 для зберігання файлів. В такому випадку, користувач через консоль AWS може створити ключ, що має обмежений доступ до хмари: тільки читання / запис його файлів в Amazon S3. Цей ключ в результаті можна застосувати для автентифікації веб-додатки в хмарі AWS(Рис.2.7).

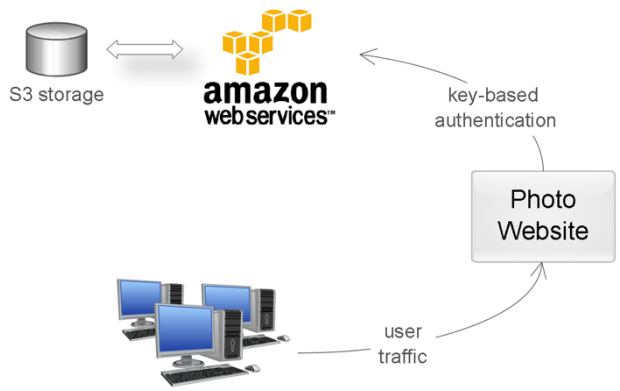

Рис.2.7 Приклад застосування автентифікації по ключу.

Використання ключів дозволяє уникнути передачі пароля користувача стороннім додаткам (в прикладі вище користувач зберіг в веб-додатку не свій пароль, а ключ доступу). Ключі мають значно більшу ентропією в порівнянні з паролями, тому їх практично неможливо підібрати. Крім того, якщо ключ був розкритий, це не призводить до компрометації основний облікового запису користувача - достатньо лише анулювати цей ключ і створити новий.

З технічної точки зору, тут не існує єдиного протоколу: ключі можуть передаватися в різних частинах HTTP-запиту: URL query, request body або HTTP header (Рис.2.8). Як і в випадку аутентифікації по паролю, найбільш оптимальний варіант - використання HTTP header. У деяких випадках використовують HTTP-схему Bearer для передачі токена в заголовку (Authorization: Bearer [token]). Щоб уникнути перехоплення ключів, з'єднання з сервером має бути обов'язково захищене протоколом SSL/TLS.

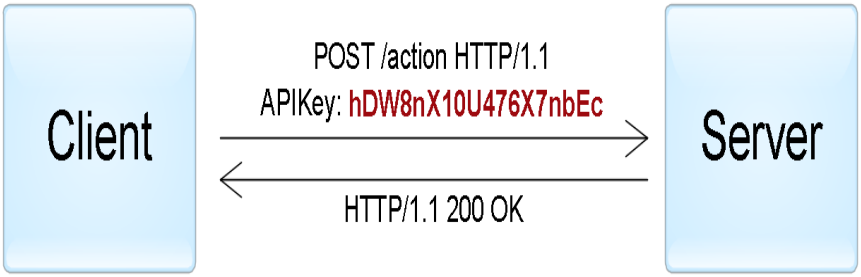

Рис.2.8 Приклад автентифікації по ключу доступу, переданого в HTTP заголовку.

Крім того, існують більш складні схеми аутентифікації по ключам для незахищених з'єднань. В цьому випадку, ключ зазвичай складається з двох частин: публічної і таємної. Публічна частина використовується для ідентифікації клієнта, а секретна частина дозволяє згенерувати підпис. Наприклад, за аналогією з digest authentication схемою, сервер може послати клієнту унікальне значення nonce або timestamp, а клієнт - повернути хеш або HMAC цього значення, обчислений з використанням секретної частини ключа.

2.5. Розгортання мережі веб-сайтів

При використанні динамічного генератора перед нами відкриваються можливості швидкого коригування вмісту десятків або сотень веб-сайтів, але для цього необхідно провести першочергову ініціалізацію та ніяк не відійти від проблеми оновлення ядра динамічного генератора для чого треба розглянути можливості використання існуючого інструментарію для даної діяльності.

У цей момент засоби управління конфігураціями і вступають в гру. У багатьох випадках, ми управляємо групами однакових серверів, на яких запущені однакові додатки і сервіси. Вони розміщуються на системах віртуалізації всередині організації, або ж запускаються як «хмарні» і гостьові в віддалених ЦОД. У деяких випадках, ми можемо говорити про велику кількість обладнання, яке існує тільки для підтримки дуже великих додатків або про обладнання, що обслуговує міріади невеликих сервісів. У будь-якому випадку, можливість «змахнути чарівною паличкою» і змусити їх усіх виконати волю системного адміністратора не може бути знецінена. Це єдиний шлях управляти величезними і зростаючими інфраструктурами.

Puppet, Chef, Ansible і Salt були задумані щоб спростити настройку та обслуговування десятків, сотень і Джае тисяч серверів. Це не означає, що маленькі компанії не отримають вигоди від цих інструментів, так як автоматизація зазвичай робить життя простіше в інфраструктурі будь-якого розміру.

Я пильно глянув на кожен з цих чотирьох інструментів, досліджував їх дизайн і функціональність, і переконаний, що незважаючи на те, що деякі оцінені вище, ніж інші, для кожного є своє місце, в залежності від цілей впровадження. Тут я підводжу підсумки моїх знахідок.

Puppet вважається найбільш використовуваним з чотирьох. Він найбільш повний з точки зору можливих дій, модулів і призначених для користувача інтерфейсів, представляючи повну картину ЦОД, охоплюючи майже кожну операційну систему і надаючи утиліти для всіх основних ОС. Початкова

установка відносно проста, вимагає розгортання головного сервера і клієнтських агентів на кожній керованої системі.

Інтерфейс командного рядка дозволяє завантажувати і встановлювати модулі за допомогою команди puppet. Потім потрібні зміни в конфігураційних файлах, необхідні для налаштування модуля під потрібні завдання та клієнти, які повинні отримати інструкції, отримають їх при наступному зверненні до головного сервера, або через запит від сервера, який ініціює процес зміни негайно.

Також є модулі, за допомогою яких виконується настройка віртуальних і розміщених в «хмарах» серверів. Всі модулі та конфігурації будуються за допомогою вбудованого, заснованого на ruby, мови, або ж на самому ruby. Це зажадає деяких знань програмування, на додаток до навичок системного адміністрування.

У Puppet Enterprise найбільш повний веб-інтерфейс з усіх, що дозволяє контролювати керовані вузли в реальному часі за допомогою попередньо створених модулів і «куховарських книг» (cookbooks), розміщених на головних серверах. Веб-інтерфейс хороший для управління, але не дозволяє проводити «тонку» настройку модулів. Інструменти для побудови звітів добре розроблені, надаючи глибоку деталізацію про поведінку агентів і про внесені зміни.

Chef схожий на Puppet з точки зору загальної концепції, в ньому також є головний сервер і агенти, встановлені на керованих вузлах. На додаток до головного сервера, установка Chef також вимагає робочої станції, для управління ним. Агенти можуть бути встановлені з робочої станції за допомогою утиліти knife, яка використовує протокол SSH для розгортання, полегшуючи тягар установки. Після цього, керовані вузли автентифіковані з головним за допомогою сертифікатів.

Конфігурація Chef тісно пов'язана з системою управління версіями Git, тому знання того, як працює Git необхідно для роботи. Також як і Puppet, Chef заснований на ruby, тому буде потрібно і знання цієї мови. Як і у випадку з

Puppet. Модулі можуть бути завантажені або написані «з нуля», після чого встановлені на управялемие вузли, відповідно до необхідних настройками.

На відміну від Puppet, у Chef поки немає добре реалізованої функції push, хоча доступна бета-версія коду. Це означає, що агентів повинні бути налаштовані на періодичну синхронізацію з головним сервером, і негайне застосування змін неможливо.

Призначений для користувача веб-інтерфейс функціональний, але не надає можливості модифікувати конфігурації. Він не так сповнений, як веб-інтерфейс Puppet Enterprise, поступається в побудові звітів і деяких інших функціях, але дозволяє вести облік обладнання та організацію вузлів.

Як і у Puppet, у Chef великий набір модулів і рецептів налаштувань, переважно на ruby. З цієї причини, Chef добре підходить для інфраструктур, орієнтованих на розробку.

Ansible більше схожий на Salt, ніж на Puppet або Chef. Ansible фокусується на оптимізації і швидкості, і не вимагає установки агентів на керовані вузли - всі функції здійснюються за SSH. Ansible написаний на python, на відміну від Puppet і Chef, заснованих на ruby.

Установка Ansible може бути виконана шляхом клонування Git-сховища на головний сервер. Слідом за цим, вузли, над якими потрібно керувати додаються в конфігурацію Ansible, і авторизовані ключі SSH «прив'язуються» до кожного вузла, ставлячись до користувача від імені якого буде запускатися Ansible. Як тільки це зроблено, головний сервер може з'єднуватися з вузлами по протоколу SSH і виконувати всі необхідні завдання. Для роботи з системами, не дозволяє доступ з правами суперкористувача (root) по SSH, Ansible використовує облікові дані, що дозволяють виконувати дії від імені суперкористувача за допомогою команди sudo.

Ansible може використовувати Paramiko - реалізацію протоколу SSH2 мовою python, або стандартну реалізацію SSH, але є також прискорений режим, для більш швидкої і широкомасштабної комунікації.

Ansible може бути запущений з командного рядка без використання конфігураційних файлів для простих завдань, таких як перевірка, що якийсь сервіс запущений, або для поновлення тригерів і перезавантаження. Для більш комплексних завдань, конфігураційні файли створюються за допомогою YAML і називаються «сценарії» (playbook). У них можуть бути використані шаблони для розширення функціональності.

У Ansible є набір модулів, які можуть використовуватися для управління різними системами, так само як і «хмарними» інфраструктурами, такими як Amazon EC2 і OpenStack. Додаткові модулі можуть бути написані на практично будь-якій мові програмування, за умови, що висновок буде в форматі JSON.

Веб-інтерфейс доступний у вигляді AnsibleWorks AWX, але безпосередньо не пов'язаний з інтерфейсом командного рядка. Це означає, що елементи конфігурації, задані через командний рядок, не з'являться в веб-інтерфейсі до тих пір, поки не буде запущений цикл синхронізації. Ви можете використовувати вбудовану утиліту для цього, але необхідно запланувати її регулярний запуск. Сам по собі веб-інтерфейсі досить функціональний, але не такий повний, як інтерфейс командного рядка, таким чином ви або будете перемикатися між обома, або ж просто скористатися командним рядком.

Salt схожий з Ansible в тому, що заснований на командному рядку. Він використовує метод push для зв'язку з клієнтами. Він може бути встановлений через Git або через систему управління пакетами на головному сервері і клієнтів. Клієнт робить запит до головного сервера, і якщо той дає дозвіл, дозволяє управляти даним вузлом за допомогою агента (в термінах Salt - minion).

Salt може зв'язуватися з клієнтами по протоколу SSH, але масштабованість значно розширюється за рахунок клієнтських агентів. Також, Salt включає асинхронний файловий сервер для прискорення обслуговування агентів, дозволяючи створювати добре масштабовані системи.

Як і в разі Ansible, ви можете віддавати команди, такі як як запуск сервісів або установка пакетів агентам за допомогою інтерпретатора, або

можете використовувати конфігураційні файли в форматі YAML (state), для обробки комплексних завдань. Також є централізовано розміщені набори даних (pillar) до яких мають доступ конфігураційні файли під час роботи.

Ви можете запросити інформацію про конфігурацію - таку як версія ядра або детальну інформацію про мережевому інтерфейсі - безпосередньо від агентів через командний рядок. Агенти можуть також задаватися через використання елементів інвентарю, званих «зернами» (grain), що дозволяють легко передавати команди на певні сервера, безвідносно до налаштованим групам. Наприклад, однією командою можна відправити запит до агентам, розташованим на серверах з певною версією ядра.

Як і попередні продукти, Salt надає велику кількість модулів для різноманітного програмного забезпечення, операційних систем і «хмарних» сервісів. Допоміжні модулі можуть бути написані на мовах python або PyDSL. Salt надає можливість керувати і Windows-вузлами, так само як і Unix, але більше розрахований на системи Unix і Linux.

Веб-інтерфейс Salt - Halite - занадто новий і не повний, як пользовательскіе інтерфейси інших систем. З його допомогою можна переглядати системні журнал повідомлень і статус керованих вузлів, а також є можливість виконувати на них команди. Цей інструмент активно розробляється, і обіцяє значні поліпшення, але поки це голий «скелет» і містить багато помилок.

Найбільша перевага Salt - масштабованість і гнучкість. У вас може бути кілька рівнів головних серверів, які організовують пов'язану струкутру, для забезпечення розподілу навантаження і збільшення відмовостійкості. Головні сервера верхнього рівня можуть управляти нижчими в ієрархії і їх підлеглими вузлами. Інша перевага - однорангова система обміну повідомленнями, що дозволяє підлеглим вузлів задавати питання головним, а ті можуть отримувати відповіді від інших серверів для завершення картини. Це може бути корисним, якщо дані для завершення налаштування вузла знаходяться в базі даних реального часу.

Що краще обрати Puppet або Chef? Ansible або Salt?

Тоді як Puppet і Chef орієнтовані на розробників, Salt і Ansible більше підходять для потреб системних адміністраторів. Простий інтерфейс і зручність використання Ansible підходять мислення системних адміністраторів в компаніях з великим числом Unix і Linux систем. Ansible швидко і легко запускається «з коробки».

Salt найнадійніший з чотирьох і теж підійде системним адміністраторам. Добре, що масштабується і володіє достатнім функціоналом, лише веб-інтерфейс залишає бажати кращого.

Puppet найбільш зрілий і, ймовірно, найдоступніший з чотирьох продукт, з точки зору зручності використання, хоча і настійно рекомендуються ґрунтовні знання ruby. Puppet не так оптимізований як Ansible або Salt, і його конфігурація часом може бути не зрозумілою Puppet найбільш безпечний для гетерогенного оточення, але ви можете порахувати Ansible або Salt більш відповідним для великих або більш гомогенних інфраструктур.

Chef стабільне і ретельно відпрацьоване рішення, але поки не дотягує до рівня Puppet з точки зору основних функцій, проте його можна розширити. Chef може бути важким для вивчення адміністраторами з недостатнім досвідом в програмуванні, але може підійти компаніям, які займаються розробкою ПЗ.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ МЕТОДИКИ ДИНАМІЧНОЇ ГЕНЕРАЦІЇ ВЕБ-САЙТІВ

3.1. Загальний план розробки динамічного генератора веб-сайтів

Відповідно раніше розглянутого підходу до динамічної генерації веб-сайту задача по створенню універсального програмного засобу здається доволі важкою та дорогою з точки зору кількості людино годин які необхідно витратити на доробку всіх необхідних класів модулів які б реалізовували додатковий функціонал. Тому було вирішено в рамках даної магістерської роботи зосередити зусилля на головній функції а саме легкозмінюванній архітектурі відображення контенту. Робочий прототип такого програмного засобу можливо розвивати у будь якому напряму включаючи як і відображення контенту так і зміну головних можливостей екземпляру генератора скажімо з умовної CMS на проксі сервер, файлове сховище, клієнт передачі запитів через мережу чи як вузол широкого мережевого сервісу. Головним чином досягнення динамічної зміни без виконання важких запитів до реляційних баз даних чи операцій з файлами на жорсткому диску хостингу може бути досягнути через використання розподіленої NoSQL бази Redis яка зберігає дані у вигляді пар ключ-значення, що було детально розглянуто у другому розділі даної роботи. Для обробки запитів та в якості інструменту роботи з Redis було б недостатньо лише самого Rack серверу який являє собою проміжний шар між Ruby програмою та веб-сервером, бо все одно виникла б необхідність у значному розширені його функціональності власними силами або через використання багатьох бібліотек. З іншого боку використання тяжкого програмного каркасу на кшталт Ruby on Rails було б зайвим через надмірну кількість додаткового коду який було б необхідно завантажувати в оперативну пам’ять серверу завантажуючи всі відповідні залежності. Тому між Rack та Ruby on Rails я зупинився на гнучкому та невеликому за об’ємом використаної пам’яті фреймворком Sintra який представляє собою набір бібліотек вкупі реалізуючи

DSL для створення API сервісу. Отже головними програмними елементами будуть Sinatra та Redis.

3.2. Середовище розробки RubyMine

«RubyMine» — комерційне інтегроване середовище розробки для розробки програмного забезпечення на Ruby та Ruby on Rails від компанії «JetBrains»(Рис.3.1). RubyMine був створений на основі платформи IntelliJ IDEA.RubyMine забезпечує інтелектуальне доповнення сирцевого коду Ruby та Ruby on Rails code, аналіз коду на льоту та підтримку рефакторингу для проектів Ruby та веб-застосунків, побудованих з Ruby on Rails. Програма підтримує популярні бібліотеки, використовувані додатків на Ruby (в тому числі Bundler, RSpec, Shoulda, Cucumber, Git).

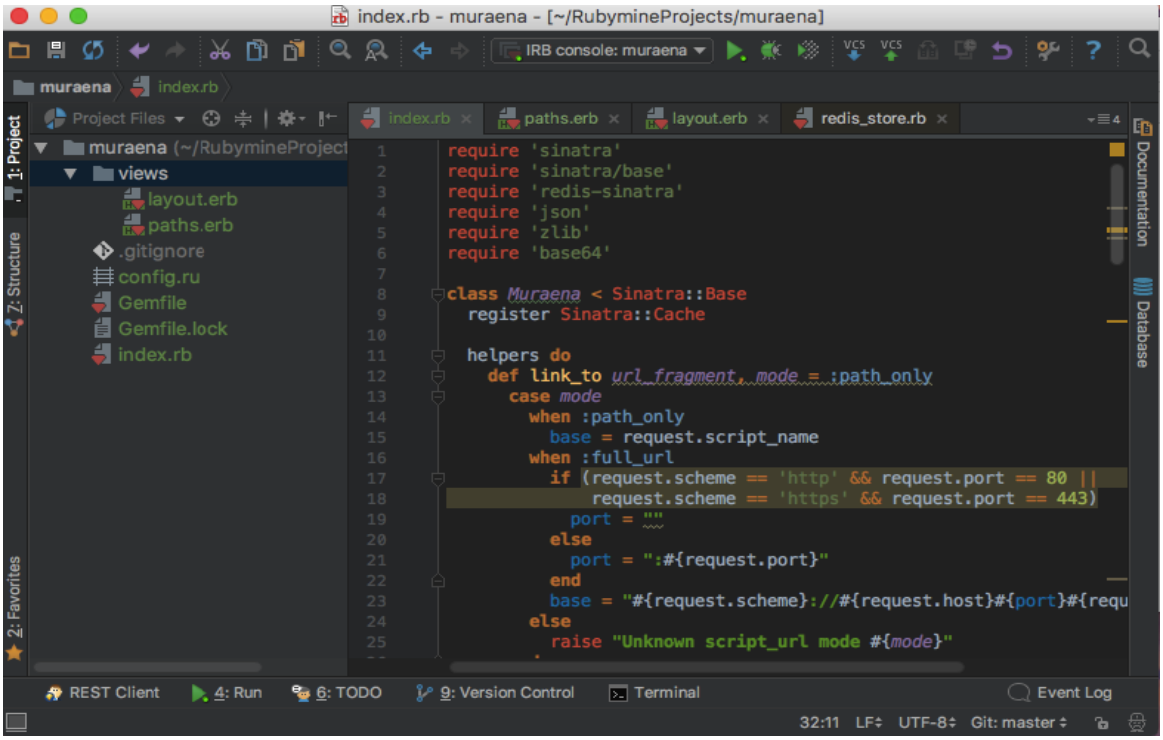

Рис.3.1 Середовище розробки RubyMine

RubyMine вигідно відрізняється від інших IDE для Ruby по ряду параметрів. По-перше, RubyMine проводить глибокий аналіз коду і здатна виводити типи

локальних змінних і типи повертаються методами значень. За рахунок цього забезпечується висока якість механізмів рефакторинга, автозавершення і валідації коду. По-друге, RubyMine підтримує безліч технологій, безпосередньо пов'язаних з Ruby-розробкою: RSpec, Cucumber, Rake, Test :: Unit, Test-Spec. Нарешті, RubyMine підтримує не тільки Ruby, але і інші мови, які використовуються при розробки веб-додатків, - HTML, CSS, JavaScript, XML

3.3. Опис роботи програмного забезпечення

Загальна схема роботи клієнту динамічного генератора доволі проста, приймаючи до сховища Redis дані клієнт прослуховує запити які надходять на його контролери і якщо в його пам’яті знаходяться дані для відповідного контролеру відображають їх користувачу. Відповідно до такої схеми основна робота по налаштуванню контенту та приведенню його в належному вигляді відбувається за межами клієнта який отримує дані ініціалізації у вигляді POST запиту(Рис.3.2).

Рис.3.2 POST-запит на ініціалізацію даних клієнта

Для виконання запиту я використвую утиліту curl у командній строці, звертаючись до ендпоінту http://localhost:9292/start. cURL — назва проекту і крос-платформового програмного засобу, що служить для передачі даних через Інтернет. cURL — це утиліта для організації вибірки даних з вебу, що надає можливість гнучкого формування запиту із завданням таких параметрів, як cookie, user_agent, referrer і будь-яких інших заголовків. cURL — це додаткова можливість оперувати з файлами на стороні сервера сторінок Інтернету за допомогою параметрів, що можуть бути переданими в рядку URL. Користуючись гнучкими можливостями командного інструменту я визначаю кодування пересланих даних як UTF-8 при типі передачі JSON. Саме тіло JSON представляє у собі хеш з одним єдиним ключем core який містить масив масивів де перший елемент являє собою адресу майбутньої сторінки, а другий елемент її закодований вміст. Частина запиту:

{"core": [["hello_page","eJyzyTC080jNyclXCEhMT7UB8gA2cQW4\n"]]}

Після ініціалізації клієнт повертає хеш який містить поле статусу та надані йому параметри для перевірки, важливо відміти що кожен виклик ініціалізатора перезаписує всі значення для маршрутів сторінок.Зайдемо на тестову сторінку та перевіримо роботу сховища даних (Рис.3.3)

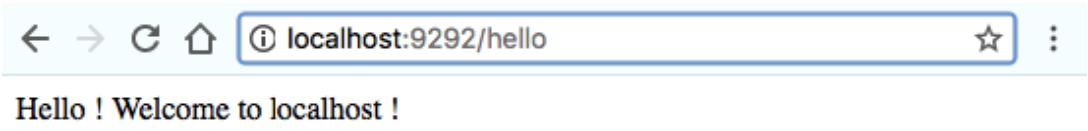

Рис.3.3 Тестова сторінка вітання

За допомогою допоміжної утиліти REST-client яка вбудована перевіримо стан ініціалізації сторінок, за допомогою окремо ендпоінту /read_by_cache_key який приймає ім’я ключа (Рис.3.4)

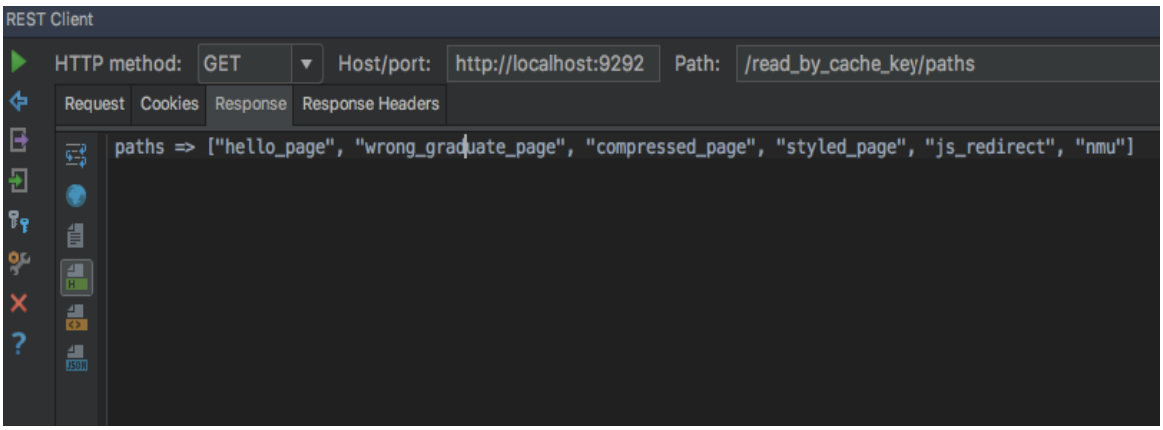

Рис.3.4 REST-client з відповіддю підтвердженням успішної ініціалізації

Як ми бачимо ініціалізація пройшла успішно і тепер можливо переглянути результати перших згенерованих сторінок.

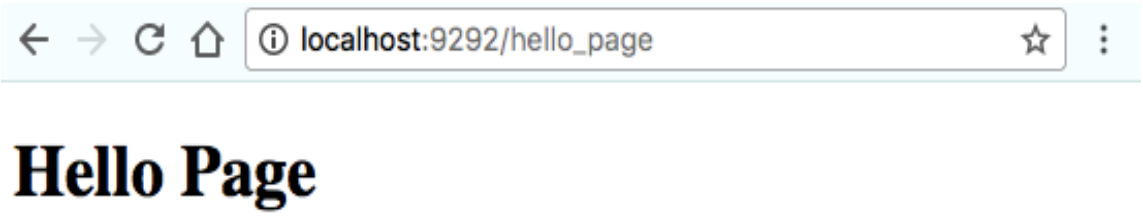

Рис.3.5 Проста сторінка з прикладом успішного відображення HTML

Коди сторінок разом з контентом зберігаються у специфічному форматі спочатку вони стискаються за домогою Ruby-обгортки над zlib, Zlib — вільна крос-платформова бібліотека для стиснення даних, яка була розроблена Жан-лу Галлі (фр. Jean-loup Gailly) та Марком Адлером (англ. Mark Adler). Є узагальненням алгоритму стиснення Deflate, який використовується у їхньому компресорі gzip. Перша публічна версія бібліотеки 0.9 була випущена 1 травня 1995 року для використання разом з бібліотекою libpng. Поширюється за ліцензією zlib. zlib є важливим компонентом багатьох програмних платформ, включаючи Linux, Mac OS X та iOS, також використаний в гральних консолях

PlayStation 3/4, Wii, Xbox 360.Потім отриманий формат за допомогою іншої обгортки над base64 переводиться у кінцевий формат в якому і зберігається, варто відмітити що Base64 — позиційна система числення з основою 64. Система широко застосовується в електронній пошті для передачі бінарних файлів у тексті листа (транспортне кодування). Всі широко відомі варіанти, відомі під назвою Base64, використовують символи A…Z, a…z і 0…9, що становить 62 знаки, для інших двох знаків в різних системах використовуються різні символи. Основа 64 (26) — це найбільший ступінь двійки, який може бути представлено лише друкованими символами ASCII. Як ми бачимо зворотне декодування на стороні серверу проходить без нарікань і помилок при транзакції вдалось уникнути(Рис.3.5, Рис.3.6)

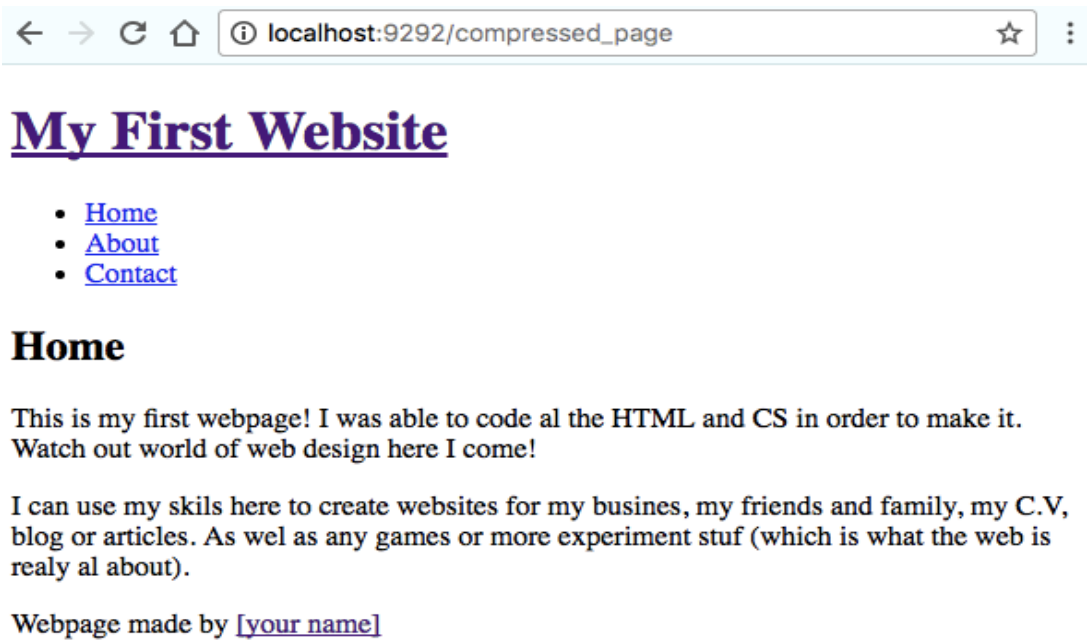

Рис.3.6 Відображення HTML сторінки з набором довільних елементів форматування

Звичайно сама лише сторінка без форматування не може складати повноцінну систему, та як ми бачимо каскадні таблиці стилів відпрацьовують також без нарікань(Рис.3.7)

Рис.3.7 HTML сторінка з вбудованими каскадними таблицями стилів

На сторінку js_redirect у закодованому вигляді був доданий код який перенаправляє користувача на офіційну сторінку органцізації об’єднаних націй, пересилання працює відмінно:

<script>window.location.replace("http://www.un.org/");</script>

Використовуючи старий трюк з фреймом у якості відображення видаленої сторінки в нашому клієнті з’являється дзеркало сайту національного гірничого університету(Рис.3.8.). Задля цього був закодований та завантажений наступний нескладний HTML код.

<iframe src="http://nmu.org.ua" height="100%" width="100%"
frameborder="0">Your browser doesnot support iframes<a href="
http://nmu.org.ua"> click here to view the page directly. </iframe>

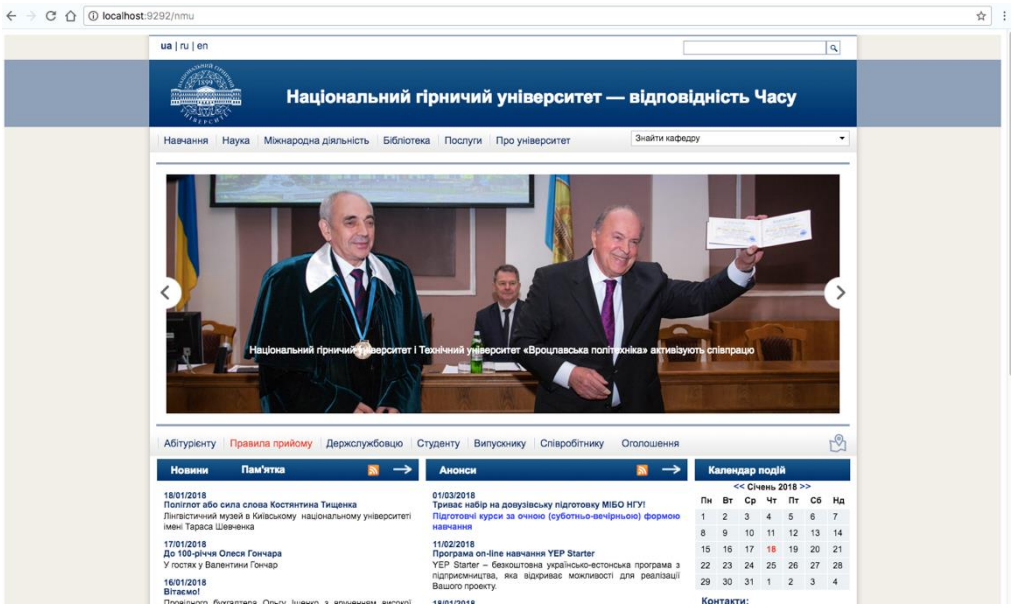

Рис.3.8 Використання фрейму для створення омани переходу на сайт національного гірничого університету

В цілому всі сторінки завантажені за допомогою ініціалізатора були успішно протестовані і вони працюють так як повинні, початкова інфраструктура була закладена і далі задля розширення функціоналу має бути проведена значна дослідницька робота та витрачено багато годин програмування на досягнення високого результату у всіх гранях відповідно до вимог представлених у теоретичних розділах.

4.ЕКОНОМІЧНА ЧАСТИНА

4.1. Визначення трудомісткості розробки програмного забезпечення

У даному дипломному проекті розглядається розробка програмного забезпечення для динамічної генерації веб-сайтів.

Нормування праці в процесі створення програмного забезпечення істотно ускладнено, якщо врахувати факт творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

1) передбачувана кількість операторів — 1000

2) коефіцієнт складності програми — 1,3

3) коефіцієнт корекції програми в ході її розробки — 0,06

4) годинна заробітна плата програміста, грн/год — 20

5) вартість машинного часу ЕОМ — 2

6) коефіцієнт кваліфікації програміста — 0.8

7) коефіцієнт збільшення витрат праці — 1.2

При розрахунку трудомісткості розробки програмного забезпечення використовується формула (4.1).

$$t = t_{тз} + t_{в} + t_{а} + t_{пр} + t_{опр} + t_{д}, \text{годин}, \quad (4.1)$$

Дана формула складається з:

$t_{о}$- витрати праці на підготовку і опис поставленого завдання (приблизно 50 люд.-годин);

$t_{и}$- витрати праці витрачені на дослідження алгоритму розв'язання задачі, люд.-годин;

$t_{а}$- витрати праці на розробку блок-схеми алгоритму, люд-годин;

$t_{п}$- витрати праці на програмування по готовій блок-схемі, люд.-годин;

$t_{отл}$- витрати праці на настроювання програми на ЕОМ, люд.-годин;

$t_{д}$ - витрати праці на підготовку документації, люд.-годин.

Одна зі складових витрат праці визначається через умовне число операторів у розробленому ПО, у число яких входять ті оператори, які необхідно написати під час роботи над програмою з урахуванням можливих уточнень в постановці завдання і вдосконалення алгоритму.

Умовне число операторів (підпрограм) визначається за допомогою формули:

$$Q = q * c\ (1 + p), \tag{4.2}$$

Формула (4.2) складається з такого набору даних:

q – передбачуване число операторів;

c – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

Підставивши значення в формулу (4.2), отримаємо:

$$Q=1000*1,3*(1+0,06) =1378$$

Витрати праці на вивчення опису завдання, з урахуванням уточнення опису і кваліфікації програміста визначаються за формулою:

$$t_в = \frac{Q * B}{(75...85) * k}\ \text{, люд.-год.} \tag{4.3}$$

де:

B – коефіцієнт збільшення витрат праці внаслідок недостатнього повного опису завдання, уточнень і деякого доопрацювання, цей коефіцієнт дорівнює 1,2.

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з данної спеціальності.

Підставимо потрібні значення в формулу (4.3), отримаємо такий результат:

$$t_в = \frac{1378 * 1.2}{80 * 0.8} = 26.8\text{, люд.-год.}$$

Витрати праці на розробку алгоритму рішення задачі, обчислюються за формулою:

$$t_a = \frac{Q}{(20...25) * k}\ \text{, люд.-год.} \tag{4.4}$$

Підставимо потрібні значення в формулу (4.4), отримаємо такий результат:

$$t_a = \frac{1378}{23*0.8} = 74.9 \text{ , люд.-год.}$$

Витрати праці на складання програми по готовій блок-схемі, обчислюються за формулою:

$$t_п = \frac{Q}{(20...25)*k} \text{, люд.-год.} \quad (4.5)$$

Підставимо потрібні значення в формулу (4.5), отримаємо такий результат:

$$t_n = \frac{1378}{20*0.8} = 86.1, \text{ люд.-год.}$$

Витрати праці на налагодження програми на ЕОМ, обчислюються за формулою:

$$t_{опр} = \frac{Q}{(4...5)*k} \text{, люд.-год.} \quad (4.6)$$

Підставимо потрібні значення в формулу (4.6), отримаємо такий результат:

$$t_{опр} = \frac{1378}{4*0.8} = 430.6, \text{ люд.-год.}$$

Витрати праці на підготовку документації по завданню, обчислюються за формулою:

$$t_д = \frac{Q}{(15...20)*k} + \frac{Q}{(15...20)*k} * 0,75 \text{ люд.-год.} \quad (4.7)$$

Розрахунок витрат на створення програмного продукту

Підставимо потрібні значення в формулу (4.8), отримаємо такий результат:

$$t_д = \frac{1378}{16*0.8} + \frac{1378}{15*0.8} = 190.4 \text{ люд.-год.}$$

отримання всіх складових формули (4.1), трудомісткість розробки програмного забезпечення складе:

t=50+26,8+74,9+86,1+430,6+190,4=858,8 люд.-год.

4.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення програмного забезпечення позначаються як $К_{по}$ включають в себе витрати на заробітну плату для розробника програми ($З_{з/п}$) і витрати машинного часу, необхідного на налагодження програми на ЕОМ ($З_{мв}$). Дані витрати визначається за формулою:

$$К_{пз} = З_{зп} + З_{мч}\text{ , грн} \quad (4.8)$$

Заробітна плата для розробника ПЗ визначається за формулою:

$$З_{зп} = t * З_{пр}\text{ , грн} \quad (4.9)$$

де:

t- загальна трудомісткість для розробника програми, люд.-год.;

$С_{пр}$- середня годинна заробітна плата програміста, грн/год.

Так як значення t ми маємо виходячи з формули (4.1), а $С_{пр}$ дорівнює приблизно 40 грн/год, то підставимо їх у формулу (4.9):

$$З_{з/п}=858,8*20= 17176 \text{ грн}$$

Значення $t_{опр}$ візьмемо з формули (4.6), а $С_{мч}$ візьмемо з формули (4.11). Розрахуємо значення за формулою (4.10):

$$Змч=430,6*2 +190,4=1091,4 \text{ грн}$$

результат отриманий з формул (4.9) та (4.10) підставимо в формулу (4.8) та розрахуємо витрати на створення програмного забезпечення:

$$Кпз=17176 +1091,4=18267.4 \text{ грн}$$

Визначені таким чином витрати на створення програмного забезпечення будуть частиною одноразових капітальних витрат на створення ПЗ.

Очікувана тривалість розробки програмного забезпечення, вираховується за формулою:

$$T = \frac{t}{B_k * F_p}\text{ , міс.} \quad (4.15)$$

де:

B_k- число розробників програмного забезпечення;

F_p - місячний фонд робочого часу (при 40-а годинному робочому тижні дорівнює 176 годин).

Знаючи число розробників, що дорівнює 1, і місячний фонд робочого часу, можемо підставити дані в формулу (4.15):

$$T = \frac{858.8}{1*176} = 5 \text{ міс.}$$

Після підрахунку, стає очевидна приблизна тривалість розробки програмного забезпечення, яка становить приблизно 5 місяців.

4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту

В даному дипломі розглянута розробка програмного забезпечення для динамічної генерації веб-сайтів. В ході дослідження своєї теми, я прийшов до висновку, що подібного програмного продукту який реалізує виключно функціонал забезпечення динамічної генерації веб-сайтів не існує. Але існують програмні продукти, зі схожим функціоналом

1. «LPgenerator»

Сервіс дозволяє створювати свій лендінг без допомоги програміста, дизайнера та верстальника в візуальному редакторі конструктора landing page - LPgenerator. Користувач може вибрати готовий шаблон для сайту, адаптований під різні бізнес-ніші. Змінювати текст і графіку на сторінці в пару кліків.

2. «AgDor»

Програма для максимальної автоматизації рутинних процесів. Дозволяє генерувати статичні та динамічні дорвеї. Впроваджувати на своїх серверах автоматичне додавання доменів і заливки. Додавати домени в панелі ISPmanager і VestaCP. Працює як на серверах так і на локальних машинах.

Порівняльна характеристика мого проекту з аналогічними продуктами.

- Продукт компанії aDvertix a.d LTD – «LPgenerator»

За рахунок цільової направленості на маркетингові функції даний продукт буде більш корисним для створення воронок продажів та оптимізації у пошукових системах ніж система яку було розроблено у магістерській роботі. Вартість ліцензії складає 5 547 грн.

Висновок: Додаток чітко створює про себе враження професійного програмного забезпечення для створення продажних сторінок

- Продукт компанії ООО «АдГор» – AgDor

Програмне забезпечення більш орієнтовано на масову генерацію сторінок з низкою якістю контенту, з великим функціоналом щодо його наповнення та перелінкування в рамках згенерованної мережі. Вартість ліцензії складає 8000 грн.

Висновок: Додаток дає змогу швидко наповнити пошукову видачу за певними ключовими словами необхідними сайтами

У порівнянні з цими продуктами, наш продукт поступається за кількістю введеного функціоналу, штату робітників і фінансуванню, однак за рахунок своєї динамічної архітектури наш продукт істотно швидше у обслуговуванні, наповненні контентом та модерації після запуску, що зможе зіграти як конкурентна перевага.

Продаж ПЗ буде застосовуватись за допомогою системи електронної торгівлі яка має бути розгорнута за допомогою самого динамічного генератора у якості саморекламної демо-версії.

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

Під час проведення маркетингового дослідження було виявлено що на ринку на разі відсутній продукт який би повністю реалізував функціонал динамічної генерації веб-сайтів. Через це важко провести точний розрахунок з інвестиційної привабливості та можливості залучення коштів на його розробку і впровадження. Також не можливо розрахувати перспективні суми грошових надходжень адже це залежить від багатьох важкопрогнозованих чинників.

Завдяки своїм головним особливостям які полягають в зменшенні кількості та тривалості операцій з розгортання та наповнення веб-сайтів впровадження ПЗ може дозволити підприємству:

- скоротити кількість працівників
- зменшити витрати на серверне устаткування
- скоротити час на адміністрування ресурсів

Висновки економічного розділу

При розрахунку трудомісткості розробки програмного забезпечення отримали значення рівні 858,8 люд.-годину. Також були розраховані витрати на створення програмного забезпечення, які склали 18 267.4 грн. Загальна тривалість розробки програмного продукту становить приблизно 5 місяців.

ВИСНОВКИ

Під час виконання магістерської роботи була створена вдосконалена методика динамічної генерації веб-сайтів яка дозволяє швидко змінювати контент на багатьох клієнтах при мінімальних затратах часу та людського ресурсу. При вирішенні поставленого завдання використовувалися наукові досягнення в областях розробки інформаційних систем та програмного забезпечення, розглядалися існуючі розробки в області генерації статичних веб-сайтів, управління контентом за допомогою систем керування вмістом.

Розроблена методика може використовуватись для вирішення широкого спектра завдань створення веб-сторінок з динамічним контентом або змінним функціональним призначенням, розгортання мереж вільного доступу до контенту та торгових сторінок.

Вдосконалена методика дозволяє створювати мережі розповсюдження контенту зі значним скороченням як матеріальних витрат, так і часових, що підтверджується розробленим програмним продуктом в даній магістерській роботі.

Для впровадження методики був створений динамічний генератор веб-сайтів з використанням мови програмування Ruby та сховища даних Redis.

У розділі «Економіка» розраховані трудомісткість розробки програмного забезпечення, витрати на створення ПО та тривалість його розробки.

Використані джерела

1. Сэм Руби, Дейв Томас, Дэвид Хэннсон - Rails 4. Гибкая разработка веб-приложений – 2014 - ООО Издательство «Питер» - 231 с.

2. Майкл Фитцджеральд - Язык программирования Ruby – 2008 – «БХВ-Петербург» - 324 с.

3. Д. Флэнаган, Ю. Мацумото - Язык программирования Ruby – 2011 - ООО Издательство «Питер» - 121 с.

4. Оби Фернандес - Путь Rails. Подробное руководство по созданию приложений в среде Ruby on Rails – 2009 – «Символ-Плюс» - 74 с.

5. Д. Томас, Д. Х. Хэнссон - Гибкая разработка веб-приложений в среде Rails – 2008 - ООО Издательство «Питер» - 342 с.

6. П. Лабберс - HTML 5 для профессионалов – 2011 – «Вильямс» - 66 с.

7. С. Уэйншенк - 100 главных принципов дизайна – 2012 – «New Riders» - 198 с.

8. Дронов В. А. - HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов – 2011 – «БХВ-Петербург» - 231 с.

9. Джон Дакетт - HTML и CSS. Разработка и дизайн веб-сайтов – 2013 – «Эксмо» - 120 с.

10. Эрик Мейер "CSS - Каскадные таблицы стилей. Подробное руководство" – 2008 - ООО Издательство «Питер» - 341 с.

11. Флэнаган Дэвид - Javascript. Подробное руководство, 6-е издание – 2012 – « Символ-Плюс» - 102 с.

12. Каслдайн Э., Шарки К. - Изучаем jQuery (2-е изд.) – 2012 - ООО Издательство «Питер» - 8 с.

13. Марк Бейтс - Programming in CoffeeScript / CoffeeScript. Второе дыхание JavaScript – 2012 – «ДМК-пресс» - 24 с.

14. Ruby on Rails – Framework [Virtual Resource] / Access Mode: URL: http://www.tutorialspoint.com/ruby-on-rails/rails-framework.htm Title from Screen

15. Ruby on Rails Tutorial by Michael Hurtl [Virtual Resource] / Access Mode: URL: http://railstutorial.ru Title from Screen

16. RSpec. Часть #1: создаем тесты для модели [Электронный ресурс] Режим доступа: URL: http://habrahabr.ru/post/53264/ Загл. с экрана.

17. Ruby on Rails API [Virtual Resource] / Access Mode: URL: http://apidock.com/rails Title from Screen

18. Devise WIKI [Virtual Resource] / Access Mode: URL: https://github.com/plataformatec/devise/wiki Title from Screen

19. Rails: хватит отмазываться пора BDD-ить! [Электронный ресурс] Режим доступа: URL: http://habrahabr.ru/post/111480/ Загл. с экрана.

20. Реализация MVC паттерна на примере создания сайта-визитки [Электронный ресурс] Режим доступа: URL: http://habrahabr.ru/post/150267/ Загл. с экрана

21. Аутентификация в Rails-приложениях с помощью Devise. Часть 1: базовая настройка [Электронный ресурс] Режим доступа: URL: http://habrahabr.ru/post/208056/ Загл. с экрана

22. Capybara API Documentation [Virtual Resource] / Access Mode: URL: http://www.rubydoc.info/github/jnicklas/capybara/master Title from Screen

23. Максим Кузнецов, Игорь Симдянов - MySQL 5 – 2010 – «БХВ-Петербург» - 400 с.

24. Omniauth API [Virtual Resource] / Access Mode: URL: https://github.com/intridea/omniauth/wiki Title from Screen

25. Pro Git Book [Virtual Resource] / Access Mode: URL http://git-scm.com/book/ru/v1 Title from Screen

26. Роутинг в Rails [Электронный ресурс] Режим доступа: URL: http://rusrails.ru/rails-routing Загл. с экрана

27. Factory Girl: Getting Started [Virtual Resource] / Access Mode: URL: https://github.com/thoughtbot/factory_girl/blob/master/GETTING_STARTED.md Title from Screen

28. Real World Example: Using factory_girl to simplify our test setup [Virtual Resource] / Access Mode: URL: http://www.agileweboperations.com/real-world-example-using-factory_girl-to-simplify-our-test-setup Title from Screen

29. 10 лучших генераторов статичных сайтов (Часть 1) [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/company/paysto/blog/289496/ Загл. с экрана

30. 10 лучших генераторов статичных сайтов (Часть 2) [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/company/paysto/blog/289504/ Загл. с экрана

31. Практическое руководство по Jekyll [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/post/207650/ Загл. с экрана

32. Что значит код как данные [Электронный ресурс] Режим доступа: URL: http://grishaev.me/code-data Загл. с экрана

33. Nginx + Lua + Redis. Эффективно обрабатываем сессию и отдаем данные [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/post/270463/ Загл. с экрана

34. Хранимые процедуры» в Redis [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/post/270251/ Загл. с экрана

35. Генераторы статических сайтов. Краткий обзор [Электронный ресурс] Режим доступа: URL: https://geekbrains.ru/posts/ssg_review Загл. с экрана

36. Мультисайтинг (многосайтовость) - это просто Краткий обзор [Электронный ресурс] Режим доступа: URL: http://www.razgonka.ru/info/39 Загл. с экрана

37. ГЕНЕРАТОРЫ СТАТИЧЕСКИХ САЙТОВ ПРОТИВ CMS: ПОБЕДИТ ЛИ ДРУЖБА?” [Электронный ресурс] Режим доступа: URL: https://webformula.pro/article/generatory-staticheskikh-saytov-protiv-cms-pobedit-li-druzhba/ Загл. с экрана

38. EVAL script numkeys key [key ...] arg [arg ...] [Virtual Resource] / Access Mode: URL: https://redis.io/commands/eval Title from Screen

39. Compressing large string in ruby [Virtual Resource] / Access Mode: URL: https://stackoverflow.com/questions/17882463/compressing-large-string-in-ruby Title from Screen

40. Подготовка серверов с помощью Chef Solo [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/post/198662/ Загл. с экрана

41. Обзор способов и протоколов аутентификации в веб-приложениях [Электронный ресурс] Режим доступа: URL: https://habrahabr.ru/company/dataart/blog/262817/ Загл. с экрана

42. 7 Methods For Tracing and Debugging Redis Lua Scripts [Virtual Resource] / Access Mode: URL: https://redislabs.com/blog/5-6-7-methods-for-tracing-and-debugging-redis-lua-scripts/ Title from Screen

43. Lua: A Guide for Redis Users [Virtual Resource] / Access Mode: URL: https://www.redisgreen.net/blog/intro-to-lua-for-redis-programmers/ Title from Screen

44. Embedding JavaScript in HTML [Virtual Resource] / Access Mode: URL: https://docstore.mik.ua/orelly/webprog/jscript/ch12_02.htm Title from Screen

45. Ruby url encoding string [Virtual Resource] / Access Mode: URL: https://stackoverflow.com/questions/6714196/ruby-url-encoding-string Title from Screen

46. Uno! Use Sinatra to Implement a REST API [Virtual Resource] / Access Mode: URL: https://www.sitepoint.com/uno-use-sinatra-implement-rest-api/ Title from Screen

47. Generating a simple links page [Virtual Resource] / Access Mode: URL:http://pythonhosted.org/htmltemplate/tutorial_1.html Title from Screen

48. zLIB [Virtual Resource] / Access Mode: https://en.wikipedia.org/wiki/Zlib Title from Screen

49. Бойко В.В.- Экономика предприятий Украины. Основной курс: Учебник для вузов. - Д.: Пороги, 1997. - 312 с.

50. Мете А.Ф., Штец К.А., Бельгольский Б.П. и др. Организация и планирование предприятий. - М.: Металлургия, 1986. - 560 с.

ДОДАТОК А

ТЕКСТ ПРОГРАМИ

Index.rb

```
require 'sinatra'
require 'sinatra/base'
require 'redis-sinatra'
require 'json'
require 'zlib'
require 'base64'

class Muraena < Sinatra::Base
  register Sinatra::Cache

  helpers do
    def link_to url_fragment, mode = :path_only
      case mode
        when :path_only
          base = request.script_name
        when :full_url
          if (request.scheme == 'http' && request.port == 80 ||
              request.scheme == 'https' && request.port == 443)
            port = ""
          else
            port = ":#{request.port}"
          end
          base =
"#{request.scheme}://#{request.host}#{port}#{request.script_name}"
        else
          raise "Unknown script_url mode #{mode}"
      end
      "#{base}#{url_fragment}"
    end

    def decoder(encoded_data)
      Zlib::Inflate.inflate(Base64.decode64(encoded_data))
    rescue
      encoded_data
    end
  end

  get '/' do
    index = settings.cache.read('index')
    unless index.nil?
      erb decoder(settings.cache.read('index'))
    else
      redirect '/hello'
    end
  end

  get '/hello' do
    settings.cache.fetch('hello') { "Hello ! Welcome to
#{request.host} !" }
```

```
  end

  get '/set_hello/:data' do
    settings.cache.write('hello', params[:data])
    { status: 'recorder' }.to_json
  end

  get '/paths' do
    @paths = settings.cache.read('paths') || []
    erb :paths
  end

  get '/:url' do
    paths = settings.cache.read('paths')
    if paths.include?(params[:url])
      erb decoder(settings.cache.read(params[:url]))
    else
      pass
    end
  end

  # Initialization
  post '/start' do
    return_message = { status: 'wrong_setup_data' }
    request.body.rewind
    jdata = JSON.parse(request.body.read, :symbolize_names =>
true)
    unless jdata.nil?
      core = jdata[:core]
      if core.is_a?(Array)
        paths = []
        core.each do |k,v|
          paths << k
          settings.cache.write('paths', paths)
          settings.cache.write(k,v)
        end
      end
      return_message[:status] = 'setup_data_accepted'
    end
    return_message[:jdata] = jdata
    return_message.to_json
  end

  # Debug section

  post '/echo' do
    request.body.rewind
    data = JSON.parse request.body.read
    "#{data}!"
  end

  get '/reset' do
    settings.cache.clear
```

```
    return_message = { status: "cache_cleared" }
    return_message.to_json
  end

  get '/read_by_cache_key/:key' do
    value = settings.cache.fetch(params[:key])
    "#{params[:key]} => #{value}"
  end

  get '/remove_by_cache_key/:key' do
    value = settings.cache.delete(params[:key])
    "#{params[:key]} => #{value}"
  end
end
```

paths.erb

```
<ul>
  <% @paths.each do |path|%>
      <li node="rep:item">
        <a href="<%=link_to(path)%>"><%= path %></a>
      </li>
  <% end %>
</ul>
```

layout.erb

```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">
</head>

<body>
  <%= yield %>
</body>
</html>
```

config.ru

```
require 'rubygems'
require 'bundler'
require 'redis-sinatra'

Bundler.require

require './index.rb'

run Muraena.new
```

Gemfile

```
source 'https://rubygems.org'
gem 'rack'
gem 'sinatra'
```

```
gem 'redis-sinatra', '~> 1.4'
gem 'html_press', '~> 0.8.2'
```

ДОДАТОК Б

ВІДГУК

на дипломну роботу магістра на тему:

«Дослідження методів динамічної генерації веб-сайтів на основі мови програмування Ruby»

студента групи 121м-16-1 Крюкова Денис Сергійовича

1. Ціль дипломної роботи магістра створення програмного забезпечення для динамічної генерації веб-сайтів.
2. Актуальність даної теми зумовлена відсутністю фреймворків з відкритим сирцевим кодом для реалізації методики .
3. Тема дипломної роботи безпосередньо зв’язана з об’єктом діяльності магістра спеціальності 8.05010301 «Програмне забезпечення систем» направлення 6.050103 «Програмна інженерія» – створення, дослідження і реалізація моделей та програмних засобів.
4. Наукова новизна результатів, які очікуються, створення нового архітектурного підходу до фреймворків базованих на принципах метапрограмування та реалізація його у вигляді програмного засобу.
5. Оригінальність технічних рішень при розробці програмного засобу полягає в створенні оригінальної методики динамічної генерації веб-сайтів на мові програмування Ruby.
6. Практична цінність дослідження полягає в розробці програмних модулів, програмного продукту, які дозволяють динамічно генерувати веб-сайти.
7. Оформлення дипломної роботи магістра виконано на сучасному рівні та відповідає вимогам, які пред'являються до робіт даної кваліфікації. Ступінь самостійності виконання достатньо висока.
8. Дипломна робота магістра в цілому заслуговує оцінки «відмінно», а сам автор – присвоєння кваліфікації «інженер-програміст».

Керівник дипломної
роботи магістра, д.т.н.,
проф. кафедри ПЗКС В.М. Куваєв

ДОДАТОК В

РЕЦЕНЗІЯ

на дипломну роботу магістра на тему:

«Дослідження методів динамічної генерації веб-сайтів на основі мови програмування Ruby»

студента групи 121м-16-1 Крюкова Денис Сергійовича

В даний час активно розвивається інтерес до мови програмування Ruby, вона широко використовується як для створення прототипів при розробці високонавантажених сервісів так і для швидкого розгортання нових бізнес додатків. Вивчення і популяризація даного інструменту вкрай важливі для того що б IT індустрія нашої держави трималася на світовому рівні. Тому обрана тема актуальна в зв'язку з ростом проникнення Ruby і суміжних технологій серед українських програмістів.

Проблеми розгортання і керування мережі веб-сайтів задля збереження і розповсюдження інформації достатньо повно висвітленні в даній роботі, а їх рішення за допомогою використання над швидкого кешування в оперативній пам’яті та змінення коду клієнту через динамічні особливості мови Ruby мають практичну значимість. Робота в цілому виконана відмінно, зауважу лише про недостатню увагу до роботи динамічного генератора в умовах великих навантажень.

Наукова новизна складається в створенні нового архітектурного підходу до фреймворків базованих на принципах метапрограмування та реалізація його у вигляді програмного засобу.

Студент Д.С. Крюков достатньо добре розібрався в специфіці використання різноманітних інформаційних технологій: Ruby, Ruby On Rails, Sinatra, Git, JS, HTML.

Використовувані технології розробки систем обробки інформації безпосередньо пов'язані з об'єктом діяльності магістра напряму 6.050103 «Програмна інженерія».

Беручи до уваги вище викладене, можна зробити висновок, що дана робота цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт рівня магістра.

З огляду на наукову новизну і ступінь опрацювання компонентів даної роботи, в цілому автор заслуговує оцінки «відмінно», а також присвоєння кваліфікації «інженер-програміст».

Професор кафедри
Автоматизації технологічних процесів
НМетАУ, д.т.н. В.І Головко