Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ «МОРСКОЙ БОЙ»

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Истомина Ильи Александровича

Научный руководитель
к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2018

ВВЕДЕНИЕ

Целью настоящей работы является разработка web-приложения, основанного на программном коде игры «Морской бой», позволяющего выполнять проверку загруженных в приложение пользовательских стратегий на языке Ruby. Для достижения поставленной цели были сформулированы следующие задачи:

— реализовать ядро web-приложения с возможностью аутентификации пользователей;
— реализовать возможность загрузки пользовательских стратегий и создания игр между ними;
— реализовать модуль приложения, ответственный за безопасную обработку игры между пользовательскими стратегиями;
— реализовать возможность графического представления обработанных игр.

В первой части работы описаны практики и технологии, на основе которых будет разработано приложение. Во второй части детально описан процесс разработки приложения и приведена демонстрация его работы.

1 Раработка Web-приложений

В данном разделе приведен обзор основных технологий и практик, необходимых для разработки web-приложения «Морской бой». Приложение разработано по принципу Модель-Представление-Контроллер на основе фреймворка Ruby on Rails.

1.1 Архитектура Web-приложений

Web-приложения представляют собой особый тип программ, построенных по архитектуре «клиент-сервер». Их особенность заключается в том, что само приложение находится и выполняется на сервере, а пользователь, обращающийся к приложению, получает только результаты его работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата. Передача запросов и результатов их обработки происходит посредством протокола HTTP [1].

Для работы пользователя с web-приложением необходимы браузер (так называемый тонкий клиент), веб-сервер (серверная часть приложения), протокол взаимодействия клиента и сервера (HTTP) и язык разметки для создания документов (HTML). В простейшем случае на сервере может быть сохранен готовый документ HTML, который по запросу будет передаваться пользователю в неизменном виде, так называемый «статический» документ. В этом случае он просто считывается с жесткого диска и передается клиенту.

В настоящее время зачастую применяется другой подход — генерация кода HTML сервером в процессе обработки запроса от клиента. Именно такой подход в построении web-приложений был применен при разработке приложения «Морской бой».

1.2 Архитектура Модель-Представление-Контроллер

Очень часто в разработке Web-приложений применяется архитектура MVC — Модель-Представление-Контроллер (Model-View-Controller). MVC — это шаблон проектирования веб-приложений, который включает в себя несколько более мелких шаблонов. При использовании MVC модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента, благодаря чему модификация одного из них оказывает минимальное воздействие на остальные или не оказывает его вовсе.

Основная цель применения MVC состоит в отделении данных и бизнес-логики от визуализации (внешнего вида). За счет такого разделения повышается возможность повторного использования программного кода и упрощается его сопровождение (изменения внешнего вида, например, не отражаются на логике приложения).

Концепция MVC разделяет данные, представление и обработку действий пользователя на компоненты:

— Модель (Model) — предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из контроллера, возвращая данные и/или изменяя своё состояние, при этом модель не содержит в себе информации, как данные можно визуализировать, а также не «общается» с пользователем напрямую;

— Представление (View) — отвечает за отображение информации (визуализацию), одни и те же данные могут представляться различными способами, например, коллекцию объектов при помощи разных представлений можно представить как в табличном виде, так и списком;

— Контроллер (Controller) — обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя, как правило, на уровне контроллера осуществляется фильтрация полученных данных и авторизация (проверяются права пользователя на выполнение действий или получение информации) [2].

Пользователь направляет запрос в контроллер (в случае веб-приложений — это обращение по адресу), контроллер обрабатывает запрос, запрашивает данные от соответствующих моделей, получает данные, может быть, выполняет какую-то дополнительную их обработку, например, агрегирует их с другими данными и затем передает данные в представление. Представление формирует данные в соответствии с заданным шаблоном отображения и возвращает результат пользователю. Это стандартная схема, по которой работает MVC-приложение.

1.3 Фреймворк Ruby on Rails

Ruby on Rails — фреймворк для веб-разработки, написанный на языке программирования Ruby. Он разработан, чтобы сделать программирование

веб-приложений проще, так как использует ряд допущений о том, что нужно каждому разработчику для создания нового проекта. Он позволяет писать меньше кода в процессе программирования, в сравнении с другими языками и фреймворками.

Философия Rails включает два важных ведущих принципа:

— Don't Repeat Yourself: DRY — это принцип разработки ПО, который гласит, что каждый фрагмент информации должен иметь единственное, неизбыточное, авторитетное представление в системе. Если придерживаться этого принципа, код будет легче поддерживать, и он будет более расширяемым и менее ошибочным;

— Convention Over Configuration: у Rails есть мнения о наилучших способах делать множество вещей в веб-приложении, и по умолчанию выставлены эти соглашения, вместо того, чтобы заставлять программиста править многочисленные конфигурационные файлы. [3]

1.3.1 Модель ActiveRecord

Active Record это реализация модели в MVC, которая является слоем в системе, ответственным за представление бизнес-логики и данных. Active Record упрощает создание и использование бизнес-объектов, данные которых требуют сохранения в базу данных. Сама по себе эта реализация шаблона Active Record является описанием системы ORM (Object Relational Mapping).

Active Record был описан Мартином Фаулером в его книге «Шаблоны архитектуры корпоративных приложений» [4]. В Active Record объекты содержат и хранимые данные, и поведение, которое работает с этими данными. Active Record исходит из мнения, что обеспечение логики доступа к данным как части объекта покажет пользователям этого объекта то, как читать и писать в базу данных.

Object Relational Mapping (объектно-реляционное отображение), обычно упоминающееся как аббревиатура ORM — это техника, соединяющая сложные объекты приложения с таблицами в системе управления реляционными базами данных. С помощью ORM, свойства и взаимоотношения этих объектов приложения могут быть сохранены и получены из базы данных без непосредственного написания выражений SQL, и, в итоге, с меньшим суммарным кодом для доступа в базу данных.

Active Record предоставляет несколько механизмов, наиболее важными из которых являются способности для:

— Представления моделей и их данных;
— Представления связей между этими моделями;
— Представления иерархий наследования с помощью связанных моделей;
— Валидации моделей до того, как они станут хранимыми в базе данных;
— Выполнения операций с базой данных в объектно-ориентированном стиле.

1.3.2 Представления ActionView

В Rails веб-запросы обрабатываются с помощью Action Controller и Action View. Обычно Action Controller ответственен за связь с базой данных и выполнение действий CRUD, тогда как Action View ответственен за компиляцию отклика на запрос.

Представления Action View пишутся с помощью тегов, содержащих специальную встроенную версию Ruby — Embedded Ruby (также ERB), смешанных с HTML. Чтобы избежать загромождения представлений шаблонным кодом, общее поведение для форм, дат и строк представлено рядом специальных классов. В терминологии Ruby on Rails они называются хелперами (от англ. helper). В существующее приложение также легко добавлять новые хелперы.

Для каждого контроллера имеется связанная директория, содержащая файлы шаблонов, которые формируют представления, связанные с этим контроллером. Эти файлы используются для отображения представления, являющегося результатом каждого метода контроллера.

1.3.3 Контроллеры ActionController

Action Controller выполняют роль контроллеров в модели MVC. После того, как роутер определит, какой контроллер использовать для обработки запроса, контроллер ответственен за осмысление запроса и генерацию подходящего ответа. В Rails Action Controller выполняет большую часть работы неявно, опираясь на соглашения, что позволяет программисту писать гораздо более простой код.

Контроллер можно рассматривать как посредник между моделями и представлениями. Он делает данные модели доступными представлению, так что оно может отобразить эти данные пользователю, а также контроллер сохраняет или обновляет данные от пользователя в модель.

Соглашение по именованию контроллеров в Rails устанавливает предпочтение множественного числа в последнем слове имени контроллера, хотя строго это не требуется (например, `ApplicationController`). К примеру, `ClientsController` более предпочтителен, чем `ClientController`.

Следование этому соглашению позволяет программисту использовать генераторы маршрутов по умолчанию и сохраняет последовательным использование хелперов URL и путей во всем приложении [5].

2 Разработка web-приложения «Морской бой»

Приложение «Морской бой» представляет собой web-сервис, отвечающий следующим требованиям:

— Приложение должно хранить информацию о пользователях, загруженных ими стратегиях, а также историю игр между стратегиями;
— Приложение должно запускать новую игру между двумя выбранными стратегиями по запросу пользователя и сохранять результат в базу данных;
— Приложение должно генерировать графическое отображение завершенной игры по запросу пользователя;
— Приложение должно сохранять стратегию пользователя в формате Ruby-кода в базу данных по запросу пользователя;
— В приложении должно быть реализовано разграничение доступа для пользователей. Так, доступ к загрузке стратегий, просмотру завершенных игр и запуску новых должен быть только у авторизованных пользователей.

Разработка приложения представляет собой реализацию функционала, связанного непосредственно со спецификой web-приложений, работой с базой данных и графическим представлением запущенных игр. Разработка игровой логики не входит в задачу, так как код, её содержащий, был предоставлен заранее и разработка приложения велась на его основе.

2.1 Программное окружение разрабатываемого приложения

Разработка данного приложения была выполнена в программном окружении, приведенном в таблице 1.

Таблица 1 – Программное окружение разрабатываемого приложения

Операционная система	Mac OS X 10.11
Язык программирования	Ruby (версия 2.5.1)
Система контроля версий	git (GitHub)
Среда разработки	RubyMine 2018.2

Также в ходе разработки приложения были использованы следующие библиотеки:

— Ruby on Rails — фреймворк для создания web-приложений на основе архитектуры MVC [6];

— Sinatra — фреймворк, предназначенный для быстрого создания максимально простых web-приложений [7];
— Devise — гибко настраиваемая библиотека для аутентификации в Rails-приложениях [8].

2.2 Разработка ядра приложения и настройка аутентификации пользователей

В данном подразделе была описана разработка основного каркаса приложения с помощью механизмов генерации кода, разработка главной страницы приложения посредством создания соответствующего контроллера и представления, а также создание модели User и внедрение механизма аутентификации.

2.3 Разработка функции загрузки пользовательских стратегий

В данном подразделе была описана разработка функционала, связанного с возможностью пользователя загружать в приложение свои стратегии в формате ruby-кода. Для этого была создана модель Strategy, описывающая объект «стратегия», создана связь между моделями User и Strategy, а также был разработан контроллер стратегий, содержащий методы для создания, просмотра и изменения пользовательских стратегий вместе с соответствующими представлениями.

2.4 Разработка функции запуска игр

В данном разделе была описана разработка функционала, связанного с запуском игр между пользовательскими стратегиями. Для этого была создана модель Game, были описаны связи между моделю Game и моделями Strategy и User, был создан контроллер игр, содержащий методы создания и просмотра завершенных игр, а также соответствующие представления.

2.5 Разработка графического отображения завершенной игры

В данном разделе была описана разработка графического отображения завершенных игр. Для этого было создано представление, содержащее элементы игрового поля на языке HTML, и был разработан скрипт на языке Javascript, отвечающий за логику воспроизведения игры и изменяющий свойства элементов поля.

2.6 Разработка модуля для исполнения пользовательского кода

В данном разделе была описана разработка модуля для исполнения пользовательских стратегий. Данный модуль был разработан на основе предоставленного кода, реализующего игру «Морской бой». Предоставленный код по большей части не подвергался изменениям, за исключением класса Game. В данный класс были внесены изменения для поддержки возможности сохранения лога игры в требуемом формате. Логика выполнения пользовательских стратегий основана на генерации исполняемых классов на основе кода стратегий, переданного в модуль в виде строки, и последующего запуска игры с этими классами и очистки ресурсов после завершения игры.

2.7 Демонстрация работы разработанного приложения

В данном разделе приведена демонстрация работы разработанного приложения. В ходе демонстрации были показаны следующие сценарии:

— Создание нового пользователя;
— Просмотр страницы профиля пользователя;
— Загрузка пользователем новой стратегии;
— Создание и запуск новой игры между стратегиями;
— Просмотр результатов завершенной игры;
— Просмотр истории всех завершенных игр.

ЗАКЛЮЧЕНИЕ

В ходе настоящей работы было разработано web-приложение, позволяющее выполнять проверку пользовательских стратегий для игры «Морской бой». Были достигнуты следующие результаты:

— была реализована возможность аутентификации пользователей а также разграничение пользовательских ролей;

— был реализован механизм загрузки пользовательских стратегий в базу приложения и запуска игр между загруженными стратегиями;

— была реализована возможность безопасной пошаговой обработки потенциально небезопасного кода стратегий путем инкапсуляции данной логики в выделенный модуль;

— было разработано графическое представление завершенных игр.

Разработанное web-приложение может быть использовано при проведении занятий по предмету «Языки программирования» на 2 курсе обучения по направлению «Программная инженерия».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Принципы работы и структура Web-приложений [Электронный ресурс]. — URL: http://www.intuit.ru/studies/courses/1139/250/lecture/6422/ (Дата обращения 01.06.2018). Загл. с экр. Яз. рус.

2 Шаблон проектирования MVC [Электронный ресурс]. — URL: https://web-creator.ru/articles/mvc (Дата обращения 01.06.2018). Загл. с экр. Яз. рус.

3 Ruby on Rails, документация на русском языке [Электронный ресурс]. — URL: http://rusrails.ru/ (Дата обращения 01.06.2018). Загл. с экр. Яз. рус.

4 *Фаулер, М.* Шаблоны корпоративных приложений / М. Фаулер. — Москва: Вильямс, 2016.

5 ActionController Overview [Электронный ресурс]. — URL: http://guides.rubyonrails.org/action_controller_overview.html (Дата обращения 01.06.2018). Загл. с экр. Яз. англ.

6 Ruby on Rails [Электронный ресурс]. — URL: http://www.rubyonrails.ru/ (Дата обращения 01.06.2018). Загл. с экр. Яз. рус.

7 Sinatra [Электронный ресурс]. — URL: http://sinatrarb.com/ (Дата обращения 01.06.2018). Загл. с экр. Яз. англ.

8 Devise [Электронный ресурс]. — URL: https://github.com/plataformatec/devise (Дата обращения 01.06.2018). Загл. с экр. Яз. англ.