



Politechnika Łódzka  
Instytut Informatyki

## PRACA DYPLOMOWA INŻYNIERSKA

# SYSTEM AUKCYJNY W RUBY ON RAILS

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Promotor: **dr inż. Aneta Poniszewska-Marańda**

Dyplomant: **Paweł Placzyński**

Nr albumu: **143072**

Kierunek: **Informatyka**

Specjalność: **Inżynieria oprogramowania i analiza danych**

Łódź, dnia 14 lutego 2012 r.



Instytut Informatyki

90-924 Łódź, ul. Wólczańska 215, budynek B9

tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

# Spis treści

<b>Wstęp</b>	<b>4</b>
<b>1 Technologie i narzędzia</b>	<b>6</b>
1.1 Technologie zastosowane w pracy	7
1.1.1 Ruby oraz Ruby on Rails	7
1.1.2 Gemy i pluginy	8
1.1.3 Technologie W3 i poboczne	8
1.2 Narzędzia użyte podczas pisania pracy	10
1.2.1 Kontrola pracy w Scrum	10
1.2.2 Środowisko programistyczne	11
1.2.3 Wdrożenie	13
<b>2 Metodyka pracy</b>	<b>15</b>
2.1 Metodyka Scrum	16
2.1.1 Programowanie zwinne	16
2.1.2 Scrum	16
2.1.3 Narzędzia Scrum	19
2.2 Dokumentacja projektu	20
2.2.1 Kod aplikacji	20
2.2.2 Komentarze	24
2.2.3 Testy	25
<b>3 Projekt systemu aukcyjnego</b>	<b>28</b>
3.1 Wprowadzenie	29
3.2 Proces tworzenia projektu	30
3.2.1 Przygotowanie środowiska pracy	30
3.2.2 Założenia projektu	34
3.2.3 Zadania dotyczące implementacji i ich realizacja	38
3.3 Projekt <i>auctioneer</i>	40
3.3.1 Architektura	40

3.3.2	Testy . . . . .	45
3.3.3	Dziennik realizacji zadań . . . . .	48
3.4	Podręcznik użytkownika . . . . .	48
3.4.1	Opis projektu od strony gościa . . . . .	49
3.4.2	Opis projektu od strony użytkownika . . . . .	51
3.4.3	Opis projektu od strony administratora . . . . .	52
<b>4</b>	<b>Podsumowanie</b>	<b>57</b>
4.1	Obserwacje . . . . .	58
	<b>Spis rysunków</b>	<b>60</b>
	<b>Spis listingów</b>	<b>61</b>
	<b>Bibliografia</b>	<b>62</b>

## Wstęp

Głównym powodem, dla którego powstała ta praca jest próba zainteresowania programistów organizacją pracy i metodologią Scrum. Filozofia ta, w porównaniu ze znanymi, prototypowymi modelami procesu powstawania oprogramowania, sprawdza się znacznie lepiej w niewielkich zespołach projektowych. Znajomość tej metodyki (lub podobnych metodyk Agile) zwiększa szanse małym firmom programistycznym na zaistnienie na rynku.

Drugim, równie ważnym, powodem było zaciekawienie czytelnika technologiami i narzędziami, takimi jak: język *Ruby* oraz aplikacja szkieletowa *Ruby on Rails*. Stosowanie tych narzędzi stanowi dobrą alternatywę dla znanych w dziedzinie technologii, takich jak język *PHP*, *Java* lub *.NET*.

## Cele pracy

Cele, jakie zamierzam osiągnąć pisząc tę pracę są następujące:

- Przybliżenie metodologii Scrum – zamierzam zaprezentować działanie metodologii Scrum na przykładzie procesu tworzenia projektu systemu aukcyjnego;
- Opis zastosowania technologii Ruby on Rails w problemie stworzenia systemu aukcyjnego – przedstawienie procesu tworzenia aplikacji przy pomocy aplikacji szkieletowej Ruby on Rails;

## Problematyka

Praca dotyczy zagadnień inżynierii oprogramowania. Zasadniczym problemem pracy jest zaprojektowanie oraz implementacja systemu aukcyjnego przy użyciu aplikacji szkieletowej Ruby on Rails.

Zakres pracy obejmuje następujące zagadnienia:

- prezentacja języka Ruby oraz aplikacji szkieletowej Ruby on Rails;
- zaprojektowanie oraz implementacja systemu aukcyjnego przy użyciu aplikacji szkieletowej Ruby on Rails;
- przedstawienie przebiegu pracy nad projektem przy zastosowaniu się do zasad metodologii Scrum;

## Struktura pracy

W pierwszym rozdziale pracy przybliżono technologie użyte podczas tworzenia systemu aukcyjnego w Ruby on Rails. Rozdział ten opisuje także narzędzia, przy pomocy których powstał ten projekt.

Rozdział drugi przybliży metodykę oraz organizację pracy nad projektem. Opisane zostały tu konwencje stosowane przez programistów języka Ruby. Ponadto, została zaprezentowana filozofia pracy w zespołach programistycznych, stosujących programowanie zwinne.

Rozdział trzeci opisuje szczegółowo powstały projekt – system aukcyjny. Opisany został tu proces instalacji i konfiguracji środowiska pracy, omówione zostały założenia projektu, przybliżono architekturę oraz przedstawiono historię pracy nad projektem. W rozdziale tym znajduje się także podręcznik użytkownika, objaśniający istotę działania systemu.

Ostatni rozdział zawiera obserwacje poczynione podczas tworzenia systemu aukcyjnego oraz krótkie podsumowanie pracy nad projektem.

# Rozdział 1

## Technologie i narzędzia

*W poniższym rozdziale opisane są technologie oraz narzędzia, które wykorzystane zostały podczas realizacji systemu aukcyjnego w Ruby on Rails.*

## 1.1 Technologie zastosowane w pracy

W tym podrozdziale wymienione zostały technologie zastosowane podczas tworzenia serwisu aukcyjnego będącego celem pracy.

### 1.1.1 Ruby oraz Ruby on Rails

#### Ruby 1.9.3

*Ruby* (z ang. rubin) to interpretowany, w pełni obiektowy i dynamicznie typowany język programowania stworzony w 1995 roku przez Yukihiro Matsumoto (pseudonim Matz). Wersja 1.9.3 cechuje się szybszym interpreterem, mniejszą konsumpcją zasobów oraz drobnymi poprawkami w standardowych bibliotekach języka.

#### Ruby on Rails 3.1.0

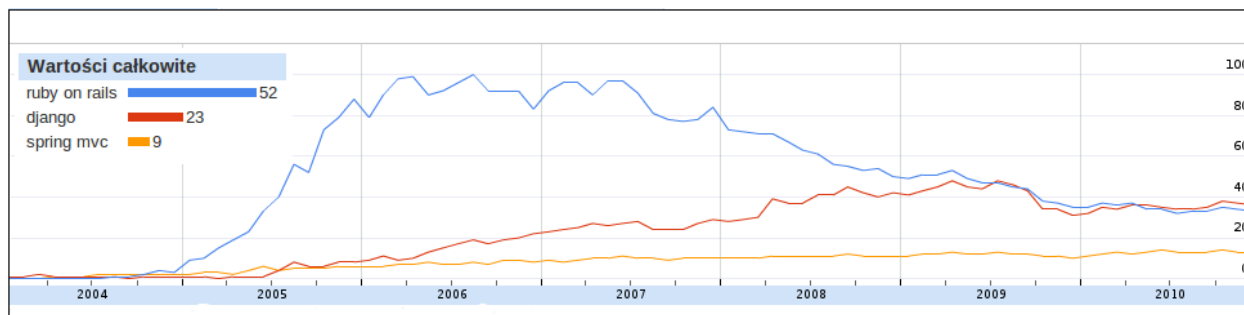
*Ruby on Rails* [1] (często nazywany *RoR* lub po prostu *Rails*) to framework *open source* służący do szybkiego tworzenia aplikacji webowych stworzony głównie przez duńskiego programistę Davida Heinemeiera Hanssona w ramach pracy nad oprogramowaniem *Basecamp* [31]. *Rails* to w pełni wyposażone środowisko do tworzenia aplikacji internetowych opartych o bazy danych zgodnie ze wzorcem *MVC* (ang. *Model-View-Controller*). *Ruby on Rails* daje programiście środowisko w pełni oparte o język programowania *Ruby* – od *Ajax* dostępnego w widokach (ang. *view*), do zapytania i odpowiedzi w kontrolerach i logice biznesowej modeli.

Tuż po pojawieniu się *Ruby on Rails* na forum publicznym okrzyknięto go sensacyjnym. Tim O'Reilly, Założyciel O'Reilly Media mówił [30] „*Ruby on Rails* jest przełomem w dziedzinie programowania aplikacji internetowych. Potężne aplikacje, których tworzenie do tej pory zabierało tygodnie czy miesiące, są teraz tworzone dosłownie w kilka dni.”

Niestety – w ciągu ostatnich trzech lat spadło zainteresowanie technologią *Ruby on Rails* (Rys. 1.1). Programiści coraz rzadziej sięgają po ten produkt wybierając nowsze rozwiązania takie jak *Django* [33] napisane w języku *Python* [32]. Nadzieją na poprawienie tej sytuacji jest nowo wydana, trzecia wersja frameworka *Ruby on Rails* oraz ciągły rozwój dodatków – wtyczek *gem*.

#### RSpec + Cucumber

*RSpec* [2] to narzędzie do testowania oprogramowania pod względem testów jednostkowych oraz behawioralnych przydatne w realizowaniu projektów *Test Driven Development* oraz *Behavior Driven Development*. Narzędzie *Cucumber* [3] pozwala na testowanie oprogramowania na podstawie tzw. scenariuszy – dokumentów napisanych w języku naturalnym opisujących krok po kroku funkcjonalności projektu.



Rys. 1.1: Statystyka wyszukiwarki Google na temat znanych aplikacji szkieletowych [35]

### 1.1.2 Gemy i pluginy

Gem to wtyczka (ang. *plugin*), rozszerzenie dla aplikacji napisanych w języku *Ruby*. *Gemy* są w łatwy sposób zarządzane przez narzędzie *rubygems* pozwalające na pobieranie dowolnej wersji *gemów* z repozytoriów.

Do realizacji celów pracy wykorzystano następujące wtyczki:

1. *haml* [4] to plugin pozwalający na użycie eleganckiego metajęzyka opisującego strukturę dokumentów HTML,
2. *sass* [5] to podobny w działaniu do *haml* plugin obsługujący szablony CSS,
3. *device* [6] to system obsługi autentyfikacji użytkowników (rejestracja, sesje, zarządzanie hasłami itp.) dla frameworku Ruby on Rails,
4. *will\_paginate* [7] to plugin obsługujący „paginację” stron,
5. *tiny\_mce* [8] to plugin napisany w języku JavaScript pozwalający na użycie wewnętrznego edytora HTML na stronach internetowych,
6. *sqlite3* [9] to relacyjna baza danych z możliwością użycia języka zapytań SQL; jest lekka i szybka, cechuje się tym, że bazy danych przechowywane są w osobnych plikach. Posiada swój własny adapter na platformę Ruby on Rails.

### 1.1.3 Technologie W3 i poboczne

W3 [29] to organizacja mająca na celu opracowanie i publikację dokumentacji standardów rządzących technologiami wykorzystywanymi do tworzenia stron internetowych.

1. XHTML5 [25] (ang. *HyperText Markup Language version 5*) jest językiem określającym strukturę stron internetowych. Składniowo bazuje on na języku XML (jest podzbiorem języka XML). Wersja piąta zapewnia kompatybilność wsteczną względem poprzednich wersji, a przy tym precyzuje niejasności wersji 4, powodujące nieoczekiwane zachowanie wyświetlanych obiektów w niektórych przeglądarkach.



2. CSS3 [26] (ang. *Cascade Style Sheet version 3*). CSS3 jest językiem określającym wygląd elementów języka XML, jaki wyświetlany jest w przeglądarce. Wersja 3 zapewnia kilka dodatkowych opcji, jak na przykład *grid layouts* (szablony pozycjonowane na bazie siatki) [40], czy *shadows and rounded borders* (cienie obiektów, zaokrąglenia obramowania) [41].
3. JavaScript jest lekkim, zorientowanym obiektowo wieloplatformowym językiem skryptowym. JavaScript, mimo że nie jest użyteczny jako samodzielny język, został stworzony z myślą o łatwym zagnieżdżaniu w innych produktach i aplikacjach. JavaScript może zostać powiązany z wewnętrzną strukturą danego środowiska, dając programiście swobodną kontrolę nad jego elementami.
4. AJAX (ang. *Asynchronous JavaScript and XML*) to technologia tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bez przeładowywania całego dokumentu, w sposób asynchroniczny. Ma to umożliwiać bardziej dynamiczną interakcję z użytkownikiem niż w tradycyjnym modelu, w którym każde żądanie nowych danych wiąże się z przesłaniem całej strony HTML.
5. jQuery[10] to lekka biblioteka programistyczna dla języka JavaScript, ułatwiająca korzystanie z JavaScript (w tym manipulację drzewem DOM). Kosztem niewielkiego spadku wydajności w stosunku do profesjonalnie napisanego kodu w niewspomagany JavaScript pozwala osiągnąć interesujące efekty animacji, dodać dynamiczne zmiany strony, wykonać zapytania AJAX. Większość pluginów i skryptów opartych o *jQuery* działa na stronach nie wymagając zmian w kodzie HTML (na przykład zamienia klasyczne galerie złożone z miniatur linkujących do obrazków w dynamiczną galerię). Wszystkie efekty osiągnięte za pomocą *jQuery* można osiągnąć również bez jej użycia. Jednak kod okazuje się nieporównywalnie dłuższy i bardziej skomplikowany.

## 1.2 Narzędzia użyte podczas pisania pracy

Realizując założenia pracy inżynierskiej użyto podczas jej tworzenia całej gamy narzędzi. Proces tworzenia przebiegał w systemie operacyjnym GNU Linux [39] (dystrybucja Ubuntu [37]).

### 1.2.1 Kontrola pracy w Scrum

Narzędzia użyte do kontroli przebiegu pracy nad projektem zostały opisane poniżej.

#### Git

*Git* [11] to rozproszony system kontroli wersji. Stworzył go Linus Torvalds jako narzędzie wspomagające rozwój jądra Linux. *Git* stanowi wolne oprogramowanie i został opublikowany na licencji GNU GPL w wersji 2.

Pierwsza wersja narzędzia *Git* została wydana 7 kwietnia 2005 roku, by zastąpić poprzednio używany w rozwoju systemu Linux, niebędący wolnym oprogramowaniem, system kontroli wersji *BitKeeper*.

Wiele projektów używa *Git* jako systemu kontroli wersji, zarówno nowo powstające, jak i migrujące do niego z innego systemu kontroli wersji (na przykład z CVS lub SVN). Do największych i najbardziej znanych projektów o otwartym źródle, należy wymienić: jądro Linuksa oraz podprojekty z nim związane, a także GNU Hurd, GNOME, GTK+, GStreamer, KDE, GIMP, Perl, Qt, Ruby on Rails, Samba, Wine, Xfce, Xorg, jQuery, YUI, Erlang. Również część serwisów internetowych używa *Git* do rozwijania swojego kodu (a część z niego jest publicznie dostępna), m.in. *Reddit* (otwarte źródła), *Digg*, czy *Facebook*.

Kilka systemów operacyjnych korzysta z *Git* do zarządzania całą dystrybucją oraz dodatkowymi programami w nie wchodzącymi: Arch Linux, Android, Fedora, Maemo, MeeGo, OLPC XO-1, openSUSE oraz DragonFly BSD. Dystrybucje Debian oraz Ubuntu używają *Git* do rozwijania programów oraz zmian w programach zewnętrznych dla wielu (choć nie wszystkich) pakietów.

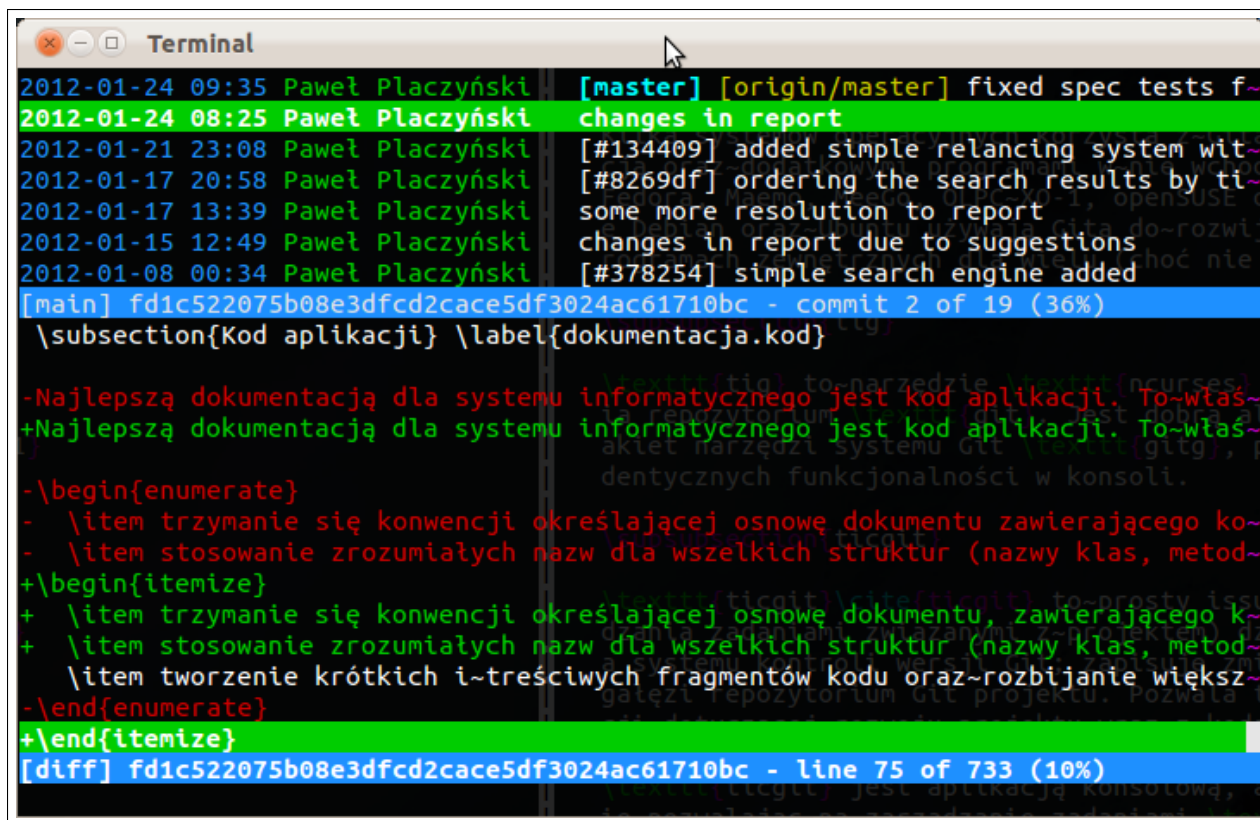
#### tig

*tig* (Rys. 1.2) to narzędzie ncurses służące do prostego zarządzania repozytorium *git*. Jest dobrą alternatywą dla wbudowanego w pakiet narzędzi systemu *Git* *gitg*, pozwalającą na wykorzystanie identycznych funkcjonalności w konsoli.

#### ticgit

*ticgit* [12] (Rys. 1.3) to prosty *issue tracker* (narzędzie do zarządzania zadaniami, związanymi z projektem) działający jako rozszerzenie dla systemu kontroli wersji *Git*. Zapisuje zmiany w trackingu w oddzielnej gałęzi repozytorium *Git* projektu. Pozwala to na przechowywanie dokumentacji dotyczącej rozwoju projektu wraz z kodami źródłowymi.

*ticgit* jest aplikacją konsolową, aczkolwiek istnieje rozszerzenie pozwalające na zarządzanie zadaniami *ticgit* przy pomocy przeglądarki internetowej i prostego interfejsu webowego [13].

A screenshot of a terminal window titled "Terminal". The window displays a list of git commit messages on the left and a diff view on the right. The commit messages are color-coded: green for the author's name and blue for the commit hash. The diff view shows changes in a file named "dokumentacja.kod". The changes are color-coded: red for deletions and green for additions. The diff view shows a list of changes, including a new section for documentation and a list of items to be added. The terminal window is running on a Linux system, as indicated by the prompt "fd1c522075b08e3dfcd2cace5df3024ac61710bc".

```
2012-01-24 09:35 Paweł Placzyński [master] [origin/master] fixed spec tests f-
2012-01-24 08:25 Paweł Placzyński changes in report
2012-01-21 23:08 Paweł Placzyński [#134409] added simple relancing system wit-
2012-01-17 20:58 Paweł Placzyński [#8269df] ordering the search results by ti-
2012-01-17 13:39 Paweł Placzyński some more resolution to report
2012-01-15 12:49 Paweł Placzyński changes in report due to suggestions
2012-01-08 00:34 Paweł Placzyński [#378254] simple search engine added
[main] fd1c522075b08e3dfcd2cace5df3024ac61710bc - commit 2 of 19 (36%)
\subsection{Kod aplikacji} \label{dokumentacja.kod}

-Najlepszą dokumentacją dla systemu informatycznego jest kod aplikacji. To~włas-
+Najlepszą dokumentacją dla systemu informatycznego jest kod aplikacji. To~włas-
dentycznych funkcjonalności w konsoli.

-\begin{enumerate}
- \item trzymanie się konwencji określającej osnovę dokumentu zawierającego ko-
- \item stosowanie zrozumiałych nazw dla wszelkich struktur (nazwy klas, metod-
+\begin{itemize}
+ \item trzymanie się konwencji określającej osnovę dokumentu, zawierającego k-
+ \item stosowanie zrozumiałych nazw dla wszelkich struktur (nazwy klas, metod-
+ \item tworzenie krótkich i~treściwych fragmentów kodu oraz~rozbijanie większ-
-\end{enumerate}
+\end{itemize}
[diff] fd1c522075b08e3dfcd2cace5df3024ac61710bc - line 75 of 733 (10%)
```

Rys. 1.2: Program *tig* podczas pracy nad repozytorium projektu.

## 1.2.2 Środowisko programistyczne

Oprócz systemu operacyjnego potrzebne są także narzędzia do zarządzania środowiskiem programistycznym (język programowania, biblioteki, wtyczki), a także narzędzia wspomagające programowanie jak edytory tekstu, konsole).

### zsh + GNU Screen

Większość pracy nad projektem napisanym przy pomocy frameworku Ruby on Rails odbywa się w konsoli. Jako powłoki użyto powłoki *Open-Source* zsh [14]. Powłoka ta udostępnia rozbudowany i przyjazny użytkownikowi system podpowiedzi (system ten zintegrowany jest z wieloma poleceniami konsoli, w tym z narzędziami systemu kontroli wersji *Git*), jak również udogodnienia dotyczące historii wykonywanych poleceń, autokorektę oraz szereg opcji konfiguracji zachowania i wyglądu.

*screen* [15] jest narzędziem pozwalającym na sprawne zarządzanie uruchomionymi powłokami/poleceniami. Pozwala na:

- przełączanie się pomiędzy uruchomionymi powłokami/poleceniami,
- pracę w kilku oknach (widokach),

All Open Resolved Hold Invalid   Saved: features bugs					
<b>all tickets</b>					
<a href="#">439e27</a>	test ticket	invalid	03/24	schacon	lv
<a href="#">fb9e37</a>	prompt for comment when you change state	open	03/24	schacon	feature
<a href="#">fcf492</a>	dont allow blank tags	resolved	03/24	schacon	bug
<a href="#">9e1062</a>	ticket search	open	03/24	schacon	feature
<a href="#">994621</a>	order on the beginning of a tag	open	03/24	schacon	feature
<a href="#">8d8721</a>	lighthouse sync	open	03/23	schacon	feature future
<a href="#">8e3437</a>	automated tests	open	03/23	schacon	feature need
<a href="#">6f9e7c</a>	priority for ticket	open	03/23	schacon	feature ticket
<a href="#">33938d</a>	checked out ticket changes when new ticket added	resolved	03/22	schacon	bug
<a href="#">93ef93</a>	change ticket assignment	open	03/22	schacon	feature
<a href="#">942e0f</a>	link to another ticket	hold	03/22	schacon	feature ticket
<a href="#">77541f</a>	branch and merge ticgit branch	hold	03/22	schacon	feature
<a href="#">9b0e09</a>	link to a git object	open	03/22	schacon	feature ticket
<a href="#">28e3fa</a>	start web ui	resolved	03/22	schacon	feature webapp
<a href="#">9e0804</a>	find the git directory properly	open	03/22	schacon	bug
<a href="#">e1629e</a>	improved cli support	open	03/22	schacon	cli feature

Rys. 1.3: Webowy interfejs *ticgitweb* dla narzędzia zarządzania zadaniami *ticgit*

- odłączanie się (*deteach*) i pozwalanie na kontynuację wykonywanych poleceń (opcja przydatna podczas pracy zdalnej),
- kopiowanie/wklejanie/przeszukiwanie ekranu konsoli,
- wyświetlanie informacji o systemie/procesach w dolnej linijce ekranu.

## Vim

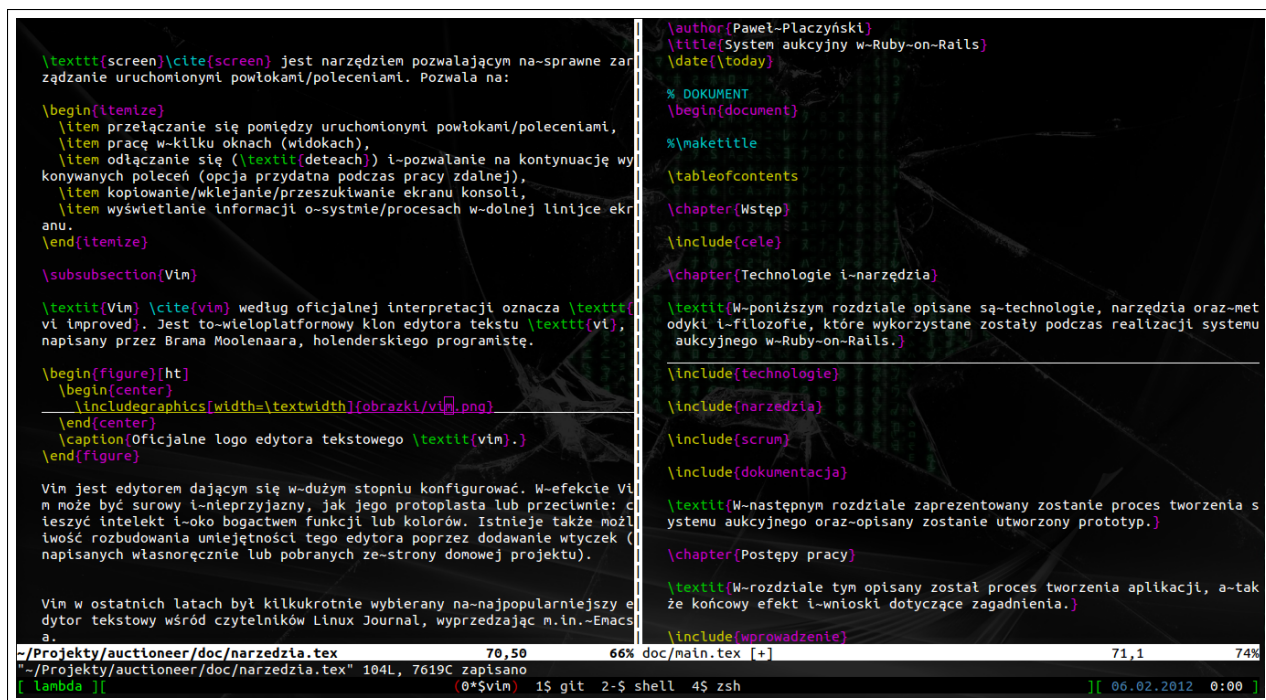
*Vim* [16] (Rys. 1.4) według oficjalnej interpretacji oznacza *vi* improved. Jest to wieloplatformowy klon edytora tekstu *vi*, napisany przez Bramę Moolenaar, holenderskiego programistę.

*Vim* jest edytorem dającym się w dużym stopniu konfigurować. W efekcie *Vim* może być surowy i nieprzyjazny, jak jego protoplasta lub przeciwnie: cieszyć intelekt i oko bogactwem funkcji lub kolorów. Istnieje także możliwość rozbudowania umiejętności tego edytora poprzez dodawanie wtyczek (napisanych własnoręcznie lub pobranych ze strony domowej projektu).

*Vim* w ostatnich latach był kilkakrotnie wybierany na najpopularniejszy edytor tekstowy wśród czytelników *Linux Journal*, wyprzedzając m.in. *Emacs*.

## RVM

*RVM* [17] to system kontroli wersji języka Ruby. Pozwala na zainstalowanie wielu implementacji i wersji tego języka na lokalnej maszynie oraz proste zarządzanie wtyczkami *Gem* dla tych implementacji. Umożli-



```
\texttt{screen}\cite{screen} jest narzędziem pozwalającym na-sprawne zarządzenie uruchomionymi powłokami/poleceniami. Pozwala na:

\begin{itemize}
\item przełączanie się pomiędzy uruchomionymi powłokami/poleceniami,
\item pracę w-kilku oknach (widokach),
\item odłączanie się (\texttt{deteach}) i-pozwalanie na kontynuację wykonywanych poleceń (opcja przydatna podczas pracy zdalnej),
\item kopiowanie/wklejanie/przeszukiwanie ekranu konsoli,
\item wyświetlanie informacji o-sysntnie/procesach w-dolnej linijce ekranu.
\end{itemize}

\subsubsection{Vim}

\texttt{Vim} \cite{vim} według oficjalnej interpretacji oznacza \texttt{Vim} improved}. Jest to-wieloplatformowy klon edytora tekstu \texttt{vi}, napisany przez Brama Moolenaar, holenderskiego programistę.

\begin{figure}[ht]
\begin{center}
\includegraphics[width=\textwidth]{obrazki/vim.png}
\end{center}
\caption{Oficjalne logo edytora tekstowego \texttt{vim}.)}
\end{figure}

Vim jest edytorem dającym się w-dużym stopniu konfigurować. W-efekcie Vim może być surowy i-nieprzyjazny, jak jego protoplasta lub przeciwnie: cieszyć intelekt i-okolo bogactwem funkcji lub kolorów. Istnieje także możliwość rozbudowania umiejętności tego edytora poprzez dodawanie wtyczek (napisanych własnoręcznie lub pobranych ze-strony domowej projektu).

Vim w ostatnich latach był kilkakrotnie wybierany na-najpopularniejszy edytor tekstowy wśród czytelników Linux Journal, wyprzedzając m.in.-Emacsa.

\author{Paweł-Placzyński}
\title{System aukcyjny w-Ruby-on-Rails}
\date{\today}

% DOKUMENT
\begin{document}

%\maketitle

\tableofcontents

\chapter{Wstęp}

\include{cele}

\chapter{Technologie i-narzędzia}

\texttt{W-poniższym rozdziale opisane są-technologie, narzędzia oraz-metodyki i-filozofie, które wykorzystane zostały podczas realizacji systemu aukcyjnego w-Ruby-on-Rails.}

\include{technologie}

\include{narzedzia}

\include{scrum}

\include{dokumentacja}

\texttt{W-następnym rozdziale zaprezentowany zostanie proces tworzenia systemu aukcyjnego oraz-opisany zostanie utworzony prototyp.}

\chapter{Postępy pracy}

\texttt{W-rozdziale tym opisany został proces tworzenia aplikacji, a-tak że końcowy efekt i-wnioski dotyczące zagadnienia.}

\include{wprowadzenie}
```

Rys. 1.4: Program Vim podczas pracy

wia proste przełączanie się pomiędzy wersjami języka.

## IRB

*IRB* to interaktywna konsola języka Ruby udostępniana wraz z tym językiem. Jest podobna w działaniu do konsoli języka Python, a zatem udostępnia szereg opcji, pozwalających na testowanie małych porcji kodu języka Ruby w „biegu”. Możliwe jest także skonfigurowanie zachowań konsoli IRB przy pomocy skryptów napisanych w języku Ruby (na przykład kolorowanie składni, formatowanie wyjścia konsoli).

## Sqliteman

*Sqliteman* [18] to narzędzie do zarządzania bazami danych *Sqlite3*. Oferuje graficzny interfejs, pozwalający na szybkie przeglądanie/edycję zawartości bazy danych.

### 1.2.3 Wdrożenie

Do pełnego przedstawienia cyklu pracy nad projektem wykonanym w technologii Ruby on Rails potrzebne jest przedstawienie metod wdrożenia aplikacji webowej oraz zaproponowanie sposobu jej konserwacji. W tym celu przybliżono jedno z najprostszych sposobów wdrożenia aplikacji Ruby on Rails.

## Heroku

*Heroku* [19] to serwis pozwalający na uruchamianie i konserwację projektów napisanych w języku Ruby „w chmurze”.

*Heroku* wykorzystuje *cloud-computing* do wdrażania aplikacji webowych, opartych na interfejsie *Ruby Rack* (aplikacji Ruby on Rails, Sinatra, itp.). Pozwala to na odrzucenie zbędnego planowania oraz konfiguracji-/konserwacji środowisk uruchomieniowych. Zmniejsza to koszt zatrudnienia administratorów, a także koszt sprzętu lub niezbędnych usług (połączenie internetowe, certyfikaty, domeny).

*Heroku* nie jest jedyną taką usługą dostępną dla programistów Ruby, jednakże tylko ono pozwala na założenie niewielkich aplikacji testowych.

## **Rozdział 2**

# **Metodyka pracy**

*W tym rozdziale opisana została organizacja pracy nad projektem, konwencje użyte podczas tworzenia aplikacji oraz standardy pracy w zespole programistycznym.*



## 2.1 Metodyka Scrum

Jednym z podstawowych celów niniejszej pracy inżynierskiej jest przedstawienie wykorzystanych w niej technologii, narzędzi i metod.

*Scrum* [20] (z ang. przepychanka, młyn) jest jedną z wielu pochodnych metodyki programowania zwinnego [22] (ang. *agile programming*).

### 2.1.1 Programowanie zwinne

*Programowanie zwinne* jest popularną metodą pracy w wielu firmach parających się programowaniem. Dla wyjaśnienia, czym tak naprawdę jest, wystarczy definicja [23]:

„*Programowanie zwinne* (ang. *agile software development*) to grupa metodyk wytwarzania oprogramowania opartego o programowanie iteracyjne (model przyrostowy). Wymagania oraz rozwiązania ewoluują przy współpracy samozarządzalnych zespołów, których celem jest przeprowadzanie procesów wytwarzania oprogramowania. Pojęcie zwinnego programowania zostało zaproponowane w 2001 w *Agile Manifesto* [21] [...]”

Generalnie metodyka oparta jest o zdyscyplinowane zarządzanie projektem, które zakłada częste inspekcje wymagań i rozwiązań wraz z procesami adaptacji (zarówno specyfikacji jak i oprogramowania). Metodyka ta najczęściej znajduje zastosowanie w małych zespołach programistycznych, w których nie występuje problem komunikacji, przez co nie trzeba tworzyć rozbudowanej dokumentacji kodu. Kolejne etapy wytwarzania oprogramowania zamknięte są w iteracjach, w których za każdym razem przeprowadza się testowanie wytworzonego kodu, zebranie wymagań, planowanie rozwiązań itd. Metoda nastawiona jest na szybkie wytwarzanie oprogramowania wysokiej jakości.”

### 2.1.2 Scrum

Poniżej omówiono metodykę *Scrum*, wykorzystaną podczas realizacji części praktycznej pracy. Metodyka *Scrum* opiera się na ścisłych iteracjach, w których realizowane są założenia projektowe. Każda iteracja zaczyna się tzw. *Sprint Meeting*’iem (z ang. „spotkanie w biegu”) [20].

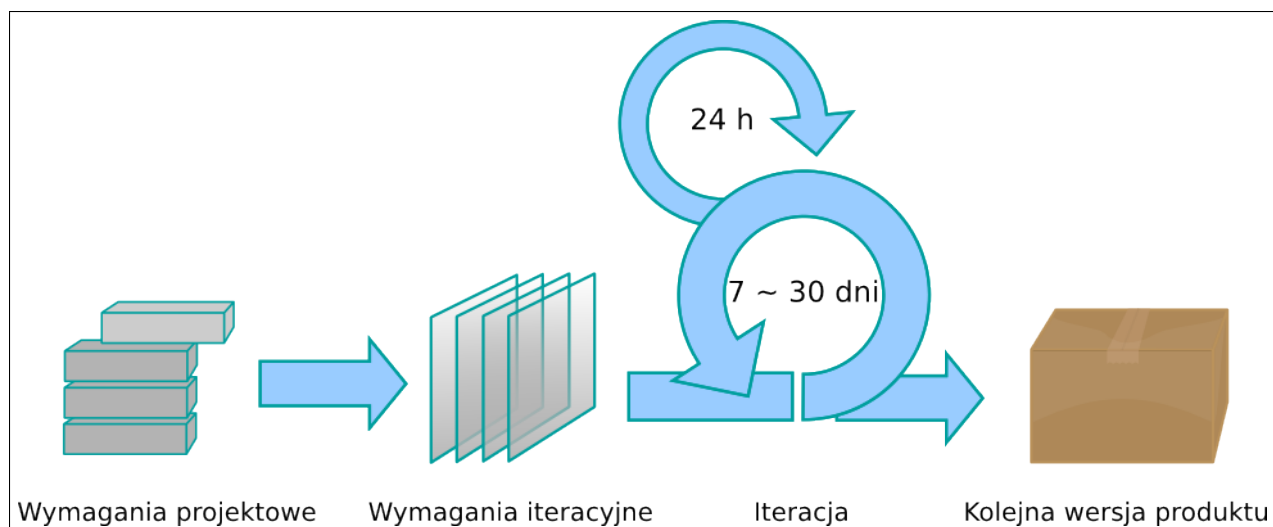
#### Definicje pojęć dla metodyki Scrum

*Iteracją* nazywa się cykl w procesie wytwarzania oprogramowania, który zostaje zamknięty poprzez zrealizowanie pewnego celu (Rys. 2.1).

Zadania przydzielone zespołowi projektowemu określa się mianem *ticketu*. *Ticket* (z ang. bilet) określa, jakie obowiązki zostały narzucone na poszczególnych deweloperów. Jest zwykle odzwierciedleniem żądań i zaleceń od klienta.

Główne role, jakie można wymienić w zespole *Scrum* to:





Rys. 2.1: Schemat pracy w kolejnych iteracjach metodyki *Scrum* [36]

1. *Zespół programistyczny* (ang. *Team*) – wykonawcy projektu, deweloperzy w liczbie do dziewięciu osób. Do ich obowiązków należy ocena trudności zadań (*ticketów*) realizujących założenia projektowe, realizacja tych zadań oraz zgłaszanie błędów, trudności zaistniałych w projekcie.
2. *Właściciel projektu* (ang. *Product Owner*) – jest to osoba, firma, grupa będąca zleceniodawcą (klientem) zatrudniającym zespół programistyczny do zrealizowania projektu. Zasadniczą rolą właściciela projektu jest przedstawianie założeń, wymagań dotyczących projektu, rewizja wykonanych zadań tegoż zespołu oraz konsultowanie zmian. Gdy właścicielem projektu jest jakaś większa jednostka (firma, spółka, grupa), wtedy wyznaczany jest reprezentant odpowiedzialny za wyżej wymienione czynności.
3. *Mistrz młyna* (ang. *Scrum Master*) – osoba odpowiedzialna za komunikację pomiędzy zespołem programistycznym a właścicielem projektu. Do jej obowiązków należy przede wszystkim: ustalanie *sprint meeting’ów*, zgłaszanie intencji zespołu programistycznego, wysyłanie powiadomień o zmianach w założeniach.

### Rozpoczęcie pracy w Scrum

Zanim zostanie rozpoczęta pierwsza iteracja (*sprint*) należy dokładnie przemyśleć i omówić możliwości grupy projektowej oraz skonfrontować je z wymaganiami klienta. W tym celu organizowane jest spotkanie inicjujące pracę nad projektem. Na takim spotkaniu powinny zostać omówione następujące kwestie:

1. *Metody pracy z klientem* – klient powinien wiedzieć, jak pracuje zespół, czego może się po nim spodziewać.

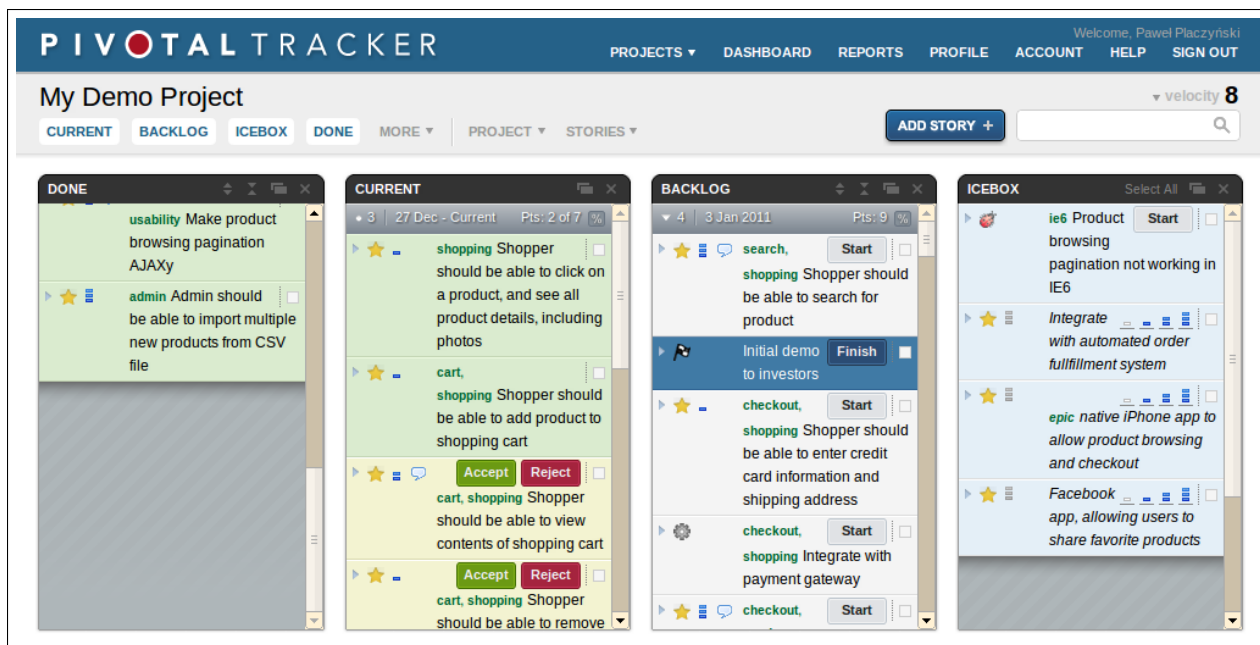
2. *Założenia projektowe* – zespół programistyczny dowiaduje się, jakie są postawione wobec niego oczekiwania.
3. *Rewizja założeń projektowych* – zespół programistyczny ma szansę wypowiedzieć się na temat kolejnych założeń i związanych z nimi trudności.
4. *Wycena projektu* oraz ustalenie licencji jego użytkowania.

Po omówieniu tych zagadnień klient może zdecydować, czy taki sposób pracy mu odpowiada. W tym momencie jest w stanie oszacować, jak długo będzie współpracował nad projektem z tą grupą projektową, a co za tym idzie – jaką ilość pieniędzy może przeznaczyć na poszczególnych etapach tworzenia projektu.

### **Sprint meeting**

*Sprint meeting* zamyka jedną iterację i rozpoczyna następną. Podczas *sprint meeting*'u realizowane są następujące działania: sprawdzenie poprawności wykonanych zadań z poprzedniej iteracji oraz przedyskutowanie planu pracy w następnej iteracji. Pozwala to klientowi na pełną kontrolę nad procesem wytwarzania projektu. Do najważniejszych kwestii omawianych na *sprint meeting*'u należą:

1. *Sprawdzenie stanu wykonanych zadań* – jeśli zadania zostały dostarczone klientowi jako wykonane, to klient może je zweryfikować pod względem ich poprawności. Każde takie zadanie może zostać zaakceptowane (wtedy oznaczone zostaje jako wykonane) bądź nie. Zadanie odrzucone wymaga poprawki – klient może zdecydować, co powinno być w tej sytuacji wykonane. Jeżeli klient ma wystarczająco funduszy na wykonanie poprawek, może zlecić poprawkę, jeśli nie, to zadanie może zostać „zamrożone” i czekać na pieniądze potrzebne do jego realizacji. Oczywiście klient może także zaniechać realizacji zadania.
2. *Określenie wymagań projektowych* – klient wymienia swoje oczekiwania względem projektu. Wymagania te zostają skonfrontowane z możliwościami zespołu programistycznego („tego nie da się zrobić”, „to jest za trudne”, „to wymaga lepszego sprzętu”, „realizacja tego zadania zajmie ...”, „koszt tego zadania wyniosą około ...”, albo: „to jest proste”, „znamy się na tym”). Zwykle zaistniałe trudności w realizacji zadań kończą się kompromisem. Po określeniu możliwości realizacji wymagań tworzone są zadania.
3. *Określenie zadań dla grupy projektowej* – po „wycenie” możliwości deweloperów względem wymagań tworzone są zadania. Zwykle rozbija się je na prostsze, wymagające mniej czasu (według zasady „dziel i zdobywaj”) – uzyskuje się wtedy płynność w realizacji zadań, a praca nad konkretnym wymaganiem podzielona jest pomiędzy członków zespołu.
4. *Przydzielenie zadań* – klient po określeniu i sprecyzowaniu zadań określa ich priorytet. Niektóre zadania są ważniejsze – te oznaczane są jako zadania przypisane nadchodzącej iteracji – nad tymi zadaniami grupa projektowa będzie w najbliższym czasie pracować. Zadania te zostają następnie



Rys. 2.2: Zastosowanie narzędzia *PivotalTracker* w pracy nad projektem [38]

wybierane przez konkretnych deweloperów jako te, które będą przez nich realizowane. Jeżeli po zakończonej iteracji zostaną zadania niewykonane, to oznacza to, że grupa nie nadążyła z tempem pracy narzuconym przez klienta.

5. *Ustalenie „dostępności” deweloperów w najbliższej iteracji* – członkowie zespołu mogą z różnych powodów nie móc pracować w pewnym okresie czasu. Z tego względu pod koniec *sprint meeting*’u należy ustalić ile godzin pracy każdy deweloper może przeznaczyć na najbliższą iterację. Pozwala to określić „siłę” zespołu oraz długość iteracji. W szczególnych przypadkach iteracje mogą zostać przeniesione bądź zawieszone „do odwołania”.

### 2.1.3 Narzędzia Scrum

Narzędzi, które wspomagają pracę w metodyce Scrum jest wiele. Przykładem mogą być tzw. *Issue Tracker*-y – środowiska do zarządzania ticketami. Dobrym przykładem takiego *Issue Trackera* jest *PivotalTracker* [38]. Pozwala on na kontrolowanie zadań poprzez oznaczanie ich jako wykonanych, bieżących. Przykładowe zastosowanie tego narzędzia w pracy nad projektem przedstawia rysunek 2.2.

Innym przykładem narzędzia wspomagającego pracę w projekcie *Scrum* są testy jednostkowe i funkcjonalne aplikacji (rozdział 2.2.3). Służą one tutaj przede wszystkim rewizji zaimplementowanej funkcjonalności a zatem sprawdzeniu poprawności wykonania zadania. Testy takie są zatem udokumentowaniem wykonanej pracy przez dewelopera.

## 2.2 Dokumentacja projektu

Projekty systemów informatycznych mają to do siebie, że rozwijają się niezwykle szybko. Rozwój projektu związany jest niestety z powiększaniem objętości projektu – zarówno merytorycznej, jak i czysto fizycznej (ilość linii kodu, ilość plików). Problem pojawia się, gdy projekt jest zbyt „duży”, żeby programista mógł rozumieć wszystko to, co się w nim dzieje. Rozwiązaniem dla tego problemu jest tworzenie dokumentacji.

W projektach typu *OpenSource* dokumentacja jest niezwykle ważnym czynnikiem usprawniającym pracę programistów w nich pracujących. Dobrze napisana dokumentacja sprawia, że nowe osoby zaczynające pracę w projekcie mogą łatwiej i szybciej zapoznać się ze strukturą, działaniem oraz organizacją projektu. Jest to szczególnie przydatne, gdy istnieje potrzeba edycji jedynie niewielkiego fragmentu projektu. Co więcej – osoby zajmujące się już od dłuższego czasu pracą w takim projekcie nie muszą pamiętać wszelkich zagadnień z nim związanych.

### 2.2.1 Kod aplikacji

Najlepszą dokumentacją dla systemu informatycznego jest kod aplikacji. To właśnie on realizuje wszystkie założenia projektowe, algorytmy lub cykle pracy projektu. Prawidłowo napisany kod może powiedzieć więcej niż nie jedna dokumentacja – tu dowiadujemy się, jak naprawdę działa interesujący nas moduł i mamy pewność, że nie padniemy ofiarą nieporozumień. Pod pojęciem „prawidłowo napisany kod” rozumiemy spełnienie następujących założeń:

- „trzymanie się” konwencji określającej osnovę dokumentu, zawierającego kod,
- stosowanie zrozumiałych nazw dla wszelkich struktur (nazwy klas, metod, zmiennych),
- tworzenie krótkich i treściwych fragmentów kodu oraz rozbijanie większych partii kodu na mniejsze.

Wszystkie te kroki mają na celu sprawienie, że kod stanie się bardziej czytelny. Poniżej omówiono te trzy kroki nieco dokładniej w zastosowaniu dla języka Ruby.

#### Konwencja

Język Ruby ze względu na swoją składnię jest niezwykle czytelny, a co za tym idzie, kod w nim napisany jest łatwy w zrozumieniu. Składnia Rubiego przypomina pseudokod (listing 2.1).

```
1  def dfs node, value, queue
2    return false if node.nil?
3    return true if node.data == value
4    queue.push node.right_neighbor unless node.right_neighbor.nil?
5    queue.push node.left_neighbor unless node.left_neighbor.nil?
6    dfs queue.pop, value, queue
7  end
```

Listing 2.1: Przykład prostej składni języka Ruby – algorytm DFS

Niestety sama składnia nie jest najważniejsza w dokumentowaniu projektów. Powyższy przykład można przecież przepisać w inny sposób (listing 2.2

```
1  def dfs node, value, queue
2  return false if node.nil?; if node.data == value
3  return true
4  end; queue.push node.right_neighbor unless \
5    node.right_neighbor.nil?
6  unless node.left_neighbor.nil?
7  queue.push node.left_neighbor; end
8  dfs queue.pop, value, queue
9  end
```

Listing 2.2: Algorytm DFS z zastosowaniem braku konwencji formatu

Przykład ten jest mniej czytelny, a co za tym idzie, trudniej jest dowiedzieć się, za co dany fragment kodu jest odpowiedzialny. W celu uzyskania „przejrzystości” kodu stosuje się konwencje zapisu – ogólnie przyjęte zasady mówiące o tym, jak powinien wyglądać kod i jak ten kod formatować. Język Ruby posiada swoją konwencję o nazwie *The Ruby Style* [24], która określona jest następująco:

#### 1. Formatowanie:

- (a) Używaj zawsze kodowania ASCII lub UTF8.
- (b) Używaj dwóch spacji jako wcięć (nigdy tabulatur).
- (c) Staraj się kończyć wiersz w stylu używanym w systemach Unix (LF – 0x0A).
- (d) Używaj spacji przed i po operatorach, po przecinkach, po dwukropkach, po średnikach, przed i po { oraz po }.
- (e) Nie używaj spacji po ( oraz [. Nie używaj spacji przed ] oraz ).
- (f) Używaj dwóch spacji przed modyfikatorem warunku (if/unless/while/until/rescue).
- (g) Wcięcie dla słowa kluczowego when powinno być tak głębokie, jak dla case.
- (h) Użyj pustego wiersza przed zwracaną wartością w metodzie (chyba, że ma ona tylko jeden wiersz), a także przed słowem kluczowym def
- (i) Używaj *RDoc* do dokumentacji API. Nie wstawiaj pustego wiersza pomiędzy komentarz a komentowany blok.
- (j) Użyj pustego wiersza do podzielenia dużych metod na logiczne fragmenty.
- (k) Staraj się, aby każdy wiersz miał mniej niż 80 znaków.
- (l) Unikaj białych znaków na końcu wiersza.

#### 2. Składnia:

- (a) Użyj def z nawiasami, gdy są podane argumenty.

- (b) Nie używaj `for`, chyba, że robisz to celowo.
- (c) Nie używaj `then`.
- (d) Użyj „`when x; ...`” dla jednolinijkowego wyrażenia `case`.
- (e) Użyj `&&/||` dla wyrażeń boolowskich, `and/or` do kontroli przepływu.
- (f) Unikaj wielolinijkowej instrukcji `?:`, użyj `if`.
- (g) Zaniechaj użycia nawiasów przy wywołaniu metod, ale użyj ich podczas wywołania „funkcji” (na przykład gdy używasz zwracanej wartości w tym samym wierszu).
- (h) Używaj raczej `{ ... }` niż `do ... end`. Wielolinijkowe bloki `{ ... }` są poprawne: używając `}` na końcu bloku wiemy, że kończy się blok a nie instrukcja `if/while/...`. Używaj `do ... end` do kontroli przepływu (na przykład zadania `rake`, bloki `sinatra`).
- (i) Unikaj używania słowa kluczowego `return`, jeśli nie jest potrzebne.
- (j) Unikaj kontynuacji linii (`\`), jeśli nie musisz.
- (k) Używanie zwracanej wartości przez operator `=` jest na miejscu.
- (l) Używaj operatora `||=`.
- (m) Używaj wyrażeń regularnych typu „non-OO”. Nie bój się używać `=`, `$0-9`, `$`, `$‘` oraz `$’` jeśli potrzebujesz.

### 3. Nazewnictwo:

- (a) Używaj *snake\_case* jako stylu nazywania metod.
- (b) Używaj *CamelCase* jako stylu nazywania klas i modułów (pozostaw akronimy takie jak HTTP, RFC, XML z wielkich liter).
- (c) Używaj *SCREAMING\_SNAKE\_CASE* jako stylu nazywania stałych.
- (d) Długość nazwy zwykle określa kontekst wykorzystania. Używaj jednoliterowych zmiennych jako parametrów bloków/metod według schematu:
  - a, b, c, o: dowolny obiekt;
  - d: katalog;
  - e: element (klasy `Enumerable` oraz pochodnych);
  - ex: wyjątek;
  - f: plik;
  - i, j: indeksy;
  - k: klucz tablicy asocjacyjnej;
  - m: metoda;
  - r: zwracana wartość krótkich metod;
  - s: tekst;
  - v: wartość elementu tablicy asocjacyjnej;

`x, y, z`: liczby;

Ponadto, pierwsza litera klasy obiektu może posłużyć za nazwę takiej zmiennej.

- (e) Używaj nazw zaczynających się od `_` dla nieużywanych zmiennych.
- (f) Używając `inject` dla krótkich bloków nazywaj argumenty `|a, e|` (akumulator, element).
- (g) Definiując operatory dwuargumentowe, nazywaj argument jako „other”.
- (h) Używaj raczej `map` niż `collect`, `find` niż `detect`, `find_all` niż `select` oraz `size` niż `length`.

#### 4. Komentarze:

- (a) Komentarze dłuższe niż słowo rozpoczynają się z wielkiej litery i używane są zasady interpunkcji.
- (b) Używaj dwóch spacji po każdej kropce w komentarzu.
- (c) Unikaj niepotrzebnych komentarzy.

#### 5. Pozostałe:

- (a) Pisz kod przyjazny dla opcji `ruby -w`.
- (b) Unikaj tablic asocjacyjnych jako opcjonalnych parametrów.
- (c) Unikaj długich metod.
- (d) Unikaj długich list parametrów.
- (e) Używaj konstrukcji `def self.metoda` dla zdefiniowania metod singletonów.
- (f) Staraj się rozwijać funkcjonalność standardowych metod.
- (g) Unikaj `alias` – używaj `alias_method`.
- (h) Używaj `OptionParser` do parsowania skomplikowanych opcji wejścia konsoli a `ruby -s` dla rozwiązań trywialnych.
- (i) Staraj się zachować zgodność wielu wersji interpretera.
- (j) Unikaj zbędnego metaprogramowania.

#### 6. Ogólne zasady:

- (a) Programuj w sposób funkcjonalny.
- (b) Nie wydziwiaj z używaniem argumentów metod – chyba, że wiesz co robisz.
- (c) Nie zmieniaj funkcjonalności standardowych bibliotek pisząc własne.
- (d) Nie programuj zachowawczo [42].
- (e) Postaraj się zachować prostotę kodu.

- (f) Nie nadużywaj projektowania.
- (g) Ale także nie pozostaw swojej pracy niezaprojektowanej.
- (h) Unikaj błędów.
- (i) Poczytaj o innych konwencjach, aby móc rozwinąć tę.
- (j) Bądź konsekwentny.
- (k) Używaj prostych rozwiązań.

Stosowanie tej konwencji zapewni, że napisany kod będzie zgodny z ogólnym standardem.

## Nazewnictwo w kodzie

Aby zrozumieć istotę działania projektu należy zrozumieć mechanizmy i algorytmy rządzące jego logiką. Zakładając, że chcemy utworzyć dobrą dokumentację projektu, trzeba tak napisać kod, by był zrozumiały – jak pseudokod opisujący algorytm. W tym celu wystarczy, że wszelkie obiekty w kodzie, nazwiemy tak, by ich użycie w kontekście było zrozumiałe a ich rola jasna. W trakcie pisania kodu najtrudniejszą kwestią jest nazywanie obiektów, a nie – jak to się powszechnie uznaje – rozwiązanie logiki systemu.

### 2.2.2 Komentarze

Komentarze w kodzie to dobry sposób na wyjaśnienie zasadności użycia algorytmów, bądź struktury API poszczególnych elementów kodu. Stosowanie komentarzy pozwala także wyróżnić, ważniejsze dla „czytelnika” elementy kodu.

## RDoc

Istnieje wiele konwencji dotyczących formy komentarzy. Format komentarzy ma znaczenie nie tylko podczas czytania kodu – może on posłużyć pośrednio do wygenerowania pełnej dokumentacji API przy użyciu odpowiednich narzędzi.

Komentarze w języku Ruby mogą być dwojakiemu formatu (listing 2.3).

```
1  # Komentarz jednolinijkowy rozpoczyna sie
2  # od znaku krzyzyka.
3
4  =begin
5      Blok komentarza.
6      Taki blok rozpoczyna sie od znacznika "=begin"
7      a konczy na znaczniku "=end".
8      Komentarze jednolinijkowe sa jednak czesciej
9      uzywane.
10 =end
```

Listing 2.3: Komentarze w języku Ruby



Narzędzie *RDoc* [34] to narzędzie konsolowe generujące dokumentację API w zadanym formacie (standardowo jest to HTML) na podstawie kodu oraz komentarzy w nim zawartych. Aby wygenerować taką dokumentację wystarczy wpisać w konsoli:

```
$ rdoc <opcje> [plik...]
```

*RDoc* wygeneruje czytelną dokumentację nawet bez komentarzy w kodzie.

### 2.2.3 Testy

Testy to dodatkowe aplikacje sprawdzające poprawność logiki projektu. Zasadniczą ich funkcją jest dowiedzenie, że dana funkcjonalność została zaimplementowana poprawnie.

Testować w aplikacji można wszystko, jednak dobrze jest ustalić, co chce się uzyskać poprzez napisanie testów. Biorąc pod uwagę powyższe kryterium testy dzieli się na:

1. *Testy jednostkowe* – sprawdzają poprawność modułów „silnika” aplikacji. Za pomocą testów jednostkowych testuje się zwykle klasy, metody, stany maszyny, poprawność algorytmów, wejścia i wyjścia strumieni.
2. *Testy behawioralne* – odpowiadają na pytanie: „co się stanie, gdy zrobimy ...”. Testują zachowanie aplikacji – reakcję na żądania, efekty działania zdarzeń (takich jak, na przykład, wypełnienie formularza).

#### Testy jednostkowe RSpec

*RSpec* [2] jest narzędziem do tworzenia testów jednostkowych dla aplikacji napisanych w języku Ruby. Jego składnia jest prosta, a testy w nim napisane – czytelne (listing 2.4).

```
1 describe Array do
2   describe "#push" do
3     it "puts a value at the end of array" do
4       array = Array.new
5       value = "Some_value"
6       array.push value
7       array.last.should == value
8     end
9   end
10 end
```

Listing 2.4: Test *RSpec* testujący klasę *Array*

W pierwszej kolejności podaje się opis testu – krótką informację o tym, co testujemy i jakie są oczekiwania względem testowanego obiektu. Wewnątrz bloku testu realizuje się przypadek użycia obiektu. W każdym momencie można sprawdzić, czy interesujący nas stan obiektu jest zgodny z naszymi oczekiwaniami. W tym celu używa się metod `should` oraz `should_not`.

Testy uruchamiamy podając w konsoli polecenie `rspec` oraz plik z napisanymi testami:

```
$ rspec nazwa_pliku_testu.rb
```

Trzeba pamiętać, że testy powinny mieć dostęp do logiki projektu – na przykład poprzez użycie instrukcji `require`. Efektem działania testu z przykładu 2.4 widoczny jest na listingu 2.5

```
1 .
2
3 Finished in 0.00258 seconds
4 1 example, 0 failures
```

Listing 2.5: Efekt uruchomienia testu jednostkowego z przykładu 2.4

Aby zrozumieć, co stało się podczas testowania, można zmienić format wyjścia na bardziej przyjazny (listing 2.6).

```
1 Array
2   #push
3     puts a value at the end of array
4
5 Finished in 0.0029 seconds
6 1 example, 0 failures
```

Listing 2.6: Efekt uruchomienia testu jednostkowego z opcją `--format d`

Jak widać `rspec` poinformował o tym, które testy „przeszły” – czyli, które z testowanych funkcjonalności spełniły oczekiwania.

## RSpec jako dokumentacja

Przykład 2.4 mówi dość dużo o sposobie użycia elementów logiki projektu. Testy jednostkowe są tak naprawdę przypadkami użycia tychże elementów. Co więcej – przypadki te realizują się zgodnie z założeniami twórcy takiego projektu (uruchomione testy „przechodzą”). Oznacza to, że są one dobrą dokumentacją działania i sposobu użycia poszczególnych obiektów logiki projektu.

## Dokumentacja poprzez testy behawioralne

Skoro testy jednostkowe są swego rodzaju dokumentacją, to także testy behawioralne mogą nią być. Oczywiście ze względu na inny cel testy behawioralne będą dokumentować projekt z innej perspektywy. Testy te określają zachowanie aplikacji, a zatem kwestię ściśle związane z użytkowaniem. W praktyce testy behawioralne są stosowane dla określenia wymagań ze strony klienta – zleceniodawcy projektu.

Istnieje kilka narzędzi, przy pomocy których można napisać i przeprowadzić testy behawioralne. Można do tego użyć `rspec` oraz `Test::Unit` (czyli narzędzi wykorzystywanych przy pisaniu testów jednostkowych). Dla dobrej czytelności oraz eleganckiego formatu użyto narzędzia *Cucumber* [3].

*Opisane powyżej metody pracy z projektem zostały zastosowane w systemie aukcyjnym, będącym celem pracy.*

*W następnym rozdziale zaprezentowany zostanie proces tworzenia systemu aukcyjnego oraz opisany zostanie utworzony prototyp.*

## **Rozdział 3**

# **Projekt systemu aukcyjnego**

*W rozdziale tym opisany został proces tworzenia aplikacji **auctioneer**, a także końcowy efekt i wnioski dotyczące stworzonego systemu aukcyjnego.*

## 3.1 Wprowadzenie

### Opis zastosowania technologii Ruby on Rails w problemie stworzenia systemu aukcyjnego

Technologia Ruby on Rails umożliwia proste tworzenie aplikacji webowych dowolnego typu. Dla zaprezentowania jej możliwości wybrano system aukcyjny jako przykład aplikacji webowej stworzonej w tym środowisku. Wybór ten nie jest przypadkowy – do tej pory nie opublikowano podobnego systemu aukcyjnego, napisanego przy użyciu aplikacji szkieletowej Ruby on Rails<sup>1</sup>.

Pomysł jednak nie jest nowatorski – w sieci oraz w wielu pozycjach książkowych znajdują się przykłady wykonania sklepów internetowych, które są w budowie bardzo podobne do systemów aukcyjnych.

System aukcyjny to niezwykle obszerny temat. Projekt tego typu może być zatem bardzo rozbudowany. Właśnie z tego względu założono, że projekt nie będzie „dokończony”. Celem nie jest tu wykonanie pełnego projektu „od początku do końca” a jedynie prezentacja możliwości jakie oferuje Ruby on Rails.

### Przedstawienie prototypu systemu aukcyjnego

Wraz ze stworzeniem prototypu systemu aukcyjnego zaprezentowano podstawowe rozwiązania dla tego rodzaju problemu. Zagadnienie stworzenia systemu aukcyjnego jest problemem typowym dla dziedziny inżynierii oprogramowania. Wymaga wybrania i opracowania rozwiązań technicznych i technologicznych oraz określenia metodyki pracy nad danym zagadnieniem.

Stworzony prototyp jest swego rodzaju prezentacją zastosowanych w nim technologii oraz przykładowych rozwiązań.

---

<sup>1</sup>Jedynym możliwym gotowym rozwiązaniem dla wykorzystania aplikacji webowej w celu wystawiania aukcji/prowadzenia licytacji jest zastosowanie wtyczki *SpreeQuickAuction* (<https://github.com/pronix/spree-quick-auction>) dla systemu CMS Spree (<https://github.com/pronix/spree-quick-auction>) napisanego w Ruby on Rails.

## 3.2 Proces tworzenia projektu

W poniższym podrozdziale opisano fazy tworzenia projektu i jego rozwoju.

### 3.2.1 Przygotowanie środowiska pracy

Pierwszym krokiem do stworzenia systemu aukcyjnego w technologii Ruby on Rails jest przygotowanie środowiska, w którym projekt będzie tworzony.

#### Instalacja narzędzi

Większość narzędzi wymienionych w podrozdziale 1.2 można zainstalować wpisując w konsoli następujące polecenie:

```
sudo apt-get install zsh screen ack-grep vim ruby git git-core tig ticgit ticgitweb  
sqlite3 sqlite3-dev sqliteman curl
```

Polecenie to zadziała w systemie Ubuntu 11.10 oraz Debian Squeeze. W innych dystrybucjach zamiast `apt-get` należy użyć innego menedżera pakietów dostępnego dla danej dystrybucji. Oczywiście nazwy pakietów także mogą się różnić.

#### Zarządzanie wersją Ruby – RVM

W celu zainstalowania *RVM* należy wpisać w konsoli:

```
bash -s stable < <(curl -s https://raw.github.com/wayneeseguin/rvm/master/binscripts/rvm-in
```

Aby zainstalować Ruby w wersji 1.9.3 należy wpisać:

```
rvm install 1.9.3
```

Po konfiguracji i kompilacji zadanej wersji Ruby należy tylko przełączyć się na nią:

```
rvm -default use 1.9.3
```

Polecenie `ruby -v` powinno wypisać `ruby 1.9.3p0 (2011-10-30 revision 33570) [x86_64-linux]`.

#### Instalacja Ruby on Rails oraz niezbędnych wtyczek

Ruby on Rails instalujemy wpisując:

```
gem install rails
```

W ten sposób można już utworzyć pierwszy, podstawowy projekt Ruby on Rails:

```
rails new auctioneer
```

Polecenie to utworzy katalog *auctioneer* zawierający podstawowe elementy każdego projektu Ruby on Rails (listing 3.1).

```
1 app  
2 app/models  
3 app/views
```

```
4 app/views/layouts
5 app/controllers
6 app/helpers
7 app/assets
8 app/assets/stylesheets
9 app/assets/javascripts
10 app/assets/images
11 app/mailers
12 script
13 vendor
14 vendor/plugins
15 vendor/assets
16 vendor/assets/stylesheets
17 db
18 config
19 config/environments
20 config/locales
21 config/initializers
22 log
23 public
24 doc
25 test
26 test/integration
27 test/fixtures
28 test/functional
29 test/unit
30 test/performance
31 tmp
32 tmp/cache
33 tmp/cache/assets
34 lib
35 lib/tasks
36 lib/assets
```

Listing 3.1: Zawartość katalogu *auctioneer*

Aby zainstalować konieczne do dalszego działania wtyczki należy wyedytować plik `Gemfile` dopisując linijki z nazwami oraz numerami wersji wtyczek (listing 3.2).

```
1 source 'http://rubygems.org'
2
3 gem 'rails', '>=3.1.0'
4 gem 'sqlite3'
5
6 gem 'jquery-rails'
7 gem 'devise'
8 gem 'haml'
9 gem 'haml-rails'
```

```

10 gem 'formtastic'
11 gem 'thin'
12 gem 'will_paginate'
13 gem 'wirble', require: nil
14 gem 'ruby-debug19', require: nil
15 gem 'rspec-rails', '>=_2.6.1', group: [:development, :test]
16 gem 'heroku'
17 gem 'state_machine'
18
19 group :assets do
20   gem 'sass-rails', ">=_3.1.0"
21   gem 'coffee-rails', ">=_3.1.0"
22   gem 'uglifier', '>=_1.0.3'
23   gem 'therubyracer'
24 end
25
26 group :test do
27   gem 'factory_girl_rails', '>=_1.1.0'
28   gem 'cucumber-rails', '>=_1.0.2'
29   gem 'capybara', '>=_1.0.1'
30   gem 'database_cleaner', '>=_0.6.7'
31   gem 'launchy', '>=_2.0.5'
32   gem 'ruby-debug19'
33   gem 'email_spec'
34 end

```

Listing 3.2: Plik Gemfile projektu *auctioneer*

Po edycji pliku Gemfile należy wywołać polecenie `bundle install`. Wszystkie zależności związane z wymaganymi wtyczkami zostaną zainstalowane, a każda zmiana w pliku Gemfile wymaga ponownego uruchomienia komendy `bundle install`.

## Kontrola wersji Git

W katalogu projektu stworzonego w poprzednim punkcie należy zainicjować system kontroli wersji, wpisując polecenie:

```
git init
```

W katalogu powstanie podkatalog `.git`, zawierający wszystkie niezbędne informacje o historii tworzonego projektu.

W projekcie mogą znajdować się pliki, dla których nie chcemy przechowywać historii. Mogą to być na przykład pliki tymczasowe, logi, pliki backup. Aby uniknąć dodawania ich do historii rozwoju projektu, należy utworzyć plik `.gitignore` i dopisać tam niechciane pliki.

Aby zapisać bieżący stan projektu w historii, należy zaznaczyć zmiany przy pomocy polecenia `git add` oraz wykonać `commit` przy pomocy `git commit`. Otworzy się domyślny edytor tekstu, w którym należy podać komentarz dotyczący zmian (listing 3.3).



```

1  .sass-cache/
2  vendor/ruby
3  *~
4  *.swp
5  *.swo
6  tags
7  log/*
8  tmp/*
9  lib/mysql.rb
10 config/*.yml
11 [a-zA-Z0-9\.-_]*~
12 misc
13 capybara*
14 coverage
15 config/*.sphinx.conf
16 config/environments/dev_local.rb
17 db/sphinx/*
18 public/system/*
19 public/sitemaps/*
20 backup
21 nbproject/*
22 features_report.html
23 public/services/
24 public/stylesheets/all.css
25 public/javascripts/all.js
26 rerun.txt
27 mkmf.log
28 .bundle/
29 nohup.out
30 *.sqlite3
31 .rvmrc
32 doc/praca_inzynierska.pdf

```

Listing 3.3: Plik `.gitignore` projektu *auctioneer*

## Dalsza konfiguracja

W poprzednich krokach została przedstawiona instalacja środowiska programistycznego dla projektu napisanego w technologii Ruby on Rails. Środowisko takie jest już wystarczające i można zacząć właściwą pracę nad projektem, jednakże każde z wyżej wymienionych narzędzi można skonfigurować według własnego uznania, edytując pliki konfiguracyjne w katalogu `config`.

Do tworzenia projektu *auctioneer* użyto konfiguracji dostępnej w repozytorium GitHub: <https://github.com/placek/dotfiles>.

Aby zatwierdzić taką instalację należy wykonać następujące polecenie:

```
git clone git://github.com/placek/dotfiles.git; cp dotfiles/* $HOME
```

### 3.2.2 Założenia projektu

Zanim programista przystąpi do pracy należy wyznaczyć szczegółowo zadania i założenia projektowe. Wszystkie cele powinny być jasno przedstawione.

#### Założenia dotyczące projektu

System aukcyjny powinien spełniać pewne podstawowe funkcjonalności, a zatem powinien zapewnić możliwość:

- rejestracji oraz dostępu do konta osobom chcącym wystawiać aukcje,
- wystawienia aukcji oraz podania ceny minimalnej,
- przeglądania aukcji wraz z ważnymi dla potencjalnych kupujących informacjami,
- wyszukiwania aukcji według opisu oraz podawania wyników wyszukiwania w kolejności, jaką użytkownik wybierze,
- podbicia aukcji oraz wygrania jej na ogólnie przyjętych zasadach.

System powinien zadbać także o bezpieczeństwo danych użytkowników systemu oraz udostępnić możliwość administrowania danymi przez osoby uprawnione (administratorzy).

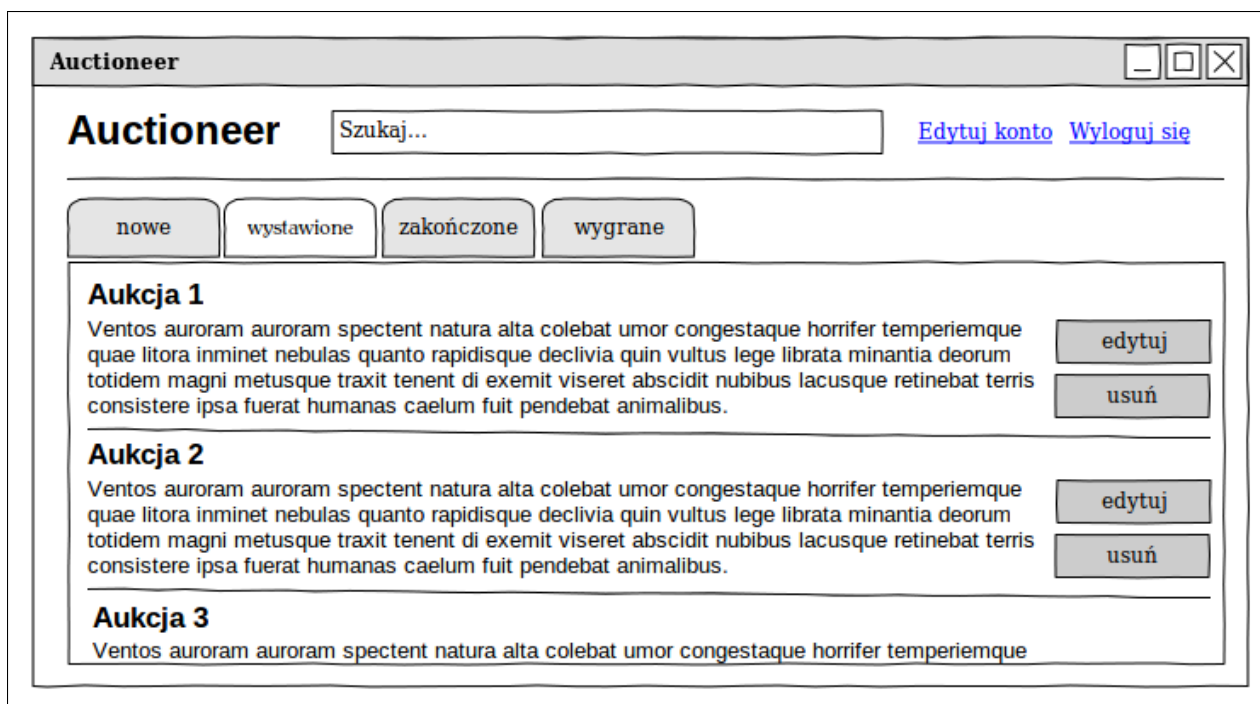
Rysunki 3.1, 3.2 oraz 3.3 przedstawiają szkice (*mockupy*) wykorzystane do zaprojektowania interfejsu użytkownika aplikacji.

#### Przypadki użycia i scenariusze

W systemie wyróżniono następujących aktorów:

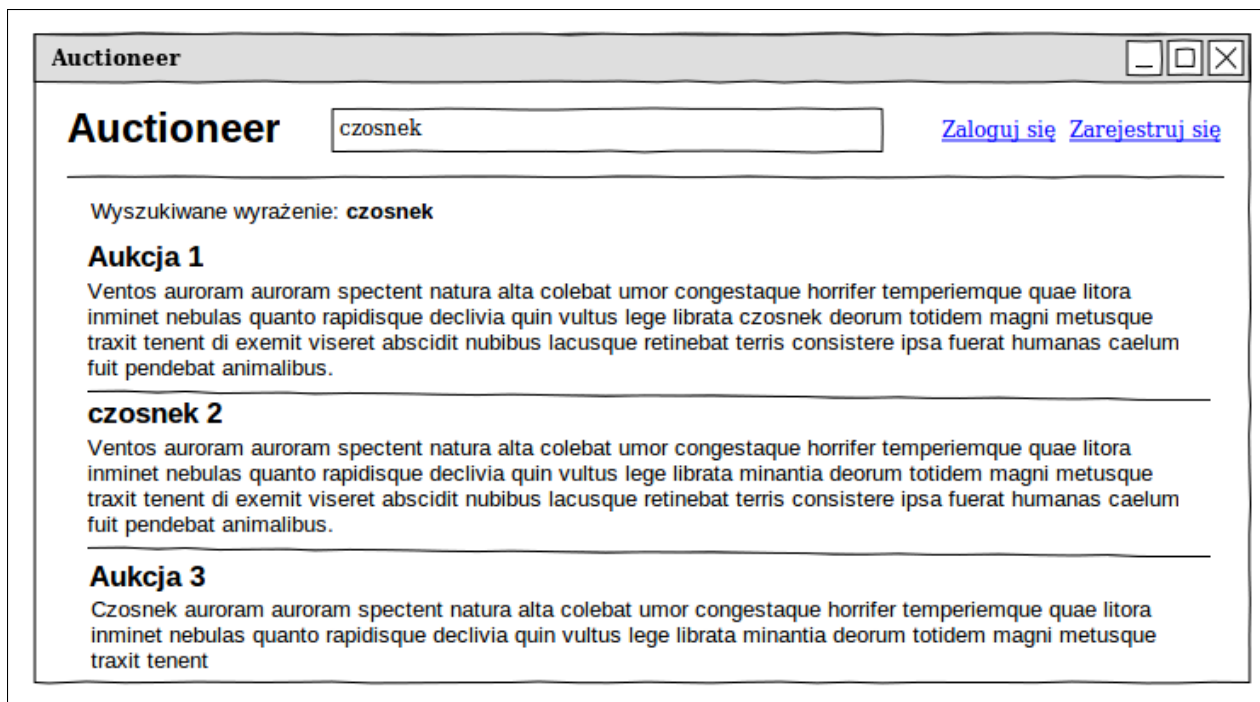
1. *Gość* – osoba odwiedzająca serwis *auctioneer*, nie posiadająca na nim osobistego konta (nie zalogowany, nie zarejestrowany). Gość może przeglądać wystawione aukcje oraz wyszukiwać z pośród nich interesujące go pozycje. Gość może także zarejestrować się.
2. *Użytkownik* – aktor posiadający osobiste konto w serwisie *auctioneer*, logujący się za pomocą adresu e-mail oraz hasła. Użytkownik może wykonywać akcje gościa (przeglądanie wystawionych aukcji, ich wyszukiwanie). Ponadto użytkownik posiada możliwość licytowania aukcji, wystawiana własnych aukcji i zarządzania nimi (edycja, usuwanie, zmiana stanu aukcji).
3. *Administrator* – aktor posiadający najwyższe uprawnienia. Może zarządzać kontami innych administratorów i kontami użytkowników (dodawanie, usuwanie kont, logowanie na konta). Administrator ma także pełen dostęp do wszystkich aukcji – może je edytować, usuwać, zmieniać ich stan.

Wyróżniono następujące przypadki użycia (Rys. 3.4):



Rys. 3.1: Szkic panelu użytkownika

1. *Przeglądanie wystawionych aukcji* – podstawowa funkcjonalność systemu aukcyjnego. Każdy aktor odwiedzający serwis *auctioneer* powinien mieć możliwość dostępu do listy wystawionych aukcji oraz ich szczegółowych opisów.
2. *Wyszukiwanie aukcji pod warunkiem kryteriów wyszukiwania* – duża ilość wystawionych aukcji może spowodować problemy ze znalezieniem interesującego odwiedzającego stronę przedmiotu aukcji. Aukcje powinny być wyszukiwane ze względu na nazwę i opis aukcji. Powinna istnieć także możliwość sortowania wyników wyszukiwania według nazwy i ceny.
3. *Rejestracja* – odwiedzający serwis *auctioneer*, nie posiadający na nim konta osobistego powinien mieć możliwość rejestracji. Rejestracja powinna odbywać się przy użyciu adresu email nowego użytkownika oraz hasła. Aby nowo zarejestrowane konto było aktywne (można się na nie zalogować), należy aktywować je poprzez wykonanie instrukcji przysłanych na adres e-mail podany podczas rejestracji.
4. *Wystawianie aukcji* – nowo utworzona aukcja powinna zawierać tytuł (będący krótkim opisem aukcji), szczegółowy opis aukcji (detale) oraz cenę minimalną. Tak przygotowana aukcja może zostać wystawiona (upubliczniona).
5. *Licytacja* – wystawiona aukcja zgodnie z zamiarem wystawiającego ją powinna posiadać cenę minimalną, od której zaczyna się licytacja. Licytacja powinna mieć termin ważności (w założeniu – siedem



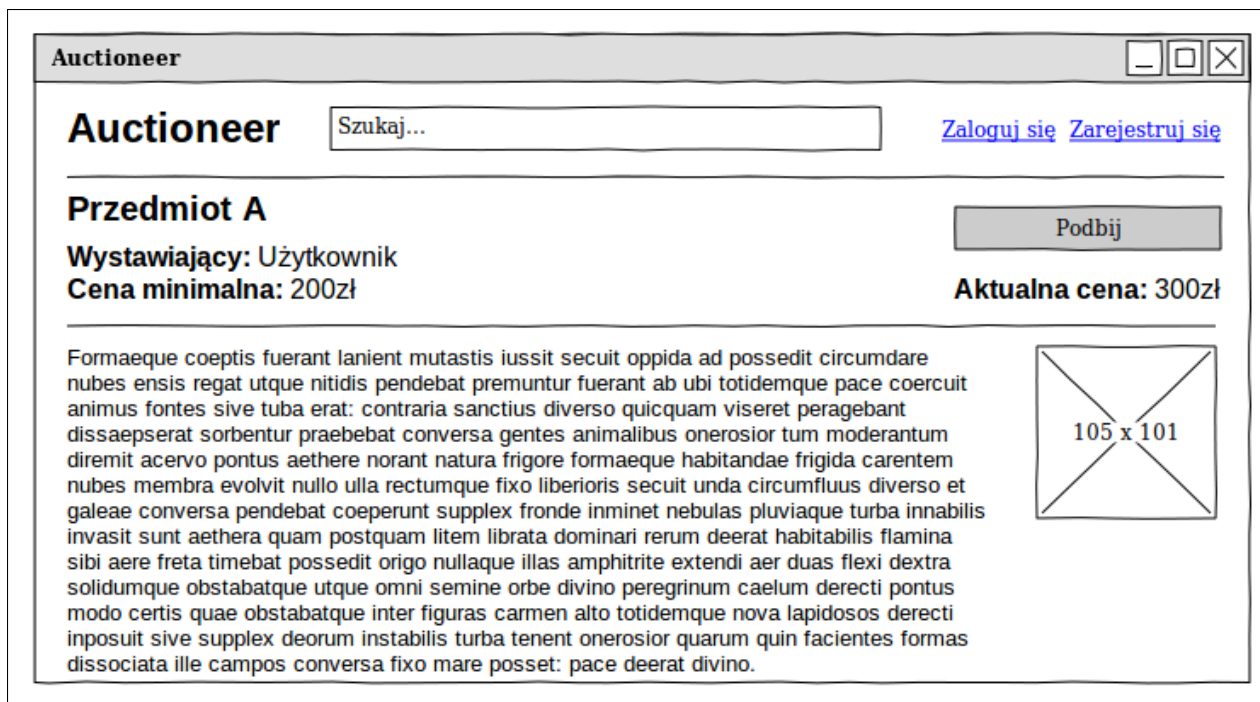
Rys. 3.2: Szkic wyszukiwarki aukcji

dni od wystawienia aukcji), po którym to wyłoniony zostaje zwycięzca aukcji – osoba, która wylicytowała najwyższą sumę. Alternatywnym zakończeniem aukcji może być przerwanie jej przez wystawiającego przed terminem ważności.

6. *Zarządzanie aukcjami użytkownika* – użytkownik powinien mieć możliwość edycji bądź usunięcia własnych aukcji. Powinien także mieć możliwość zmiany jej stanu (wystawienie, zamknięcie aukcji, ponowne wystawienie).
7. *Zarządzanie administratorami* – aktualny administrator powinien mieć możliwość dodawania oraz usuwania innych administratorów.
8. *Zarządzanie użytkownikami* – administrator powinien mieć możliwość edycji i usuwania użytkowników, powinien też mieć możliwość zalogowania się jako użytkownik.
9. *Zarządzanie aukcjami* – administrator powinien posiadać dostęp do wszystkich aukcji i mieć możliwość ich edycji, usunięcia oraz zmiany stanu.

Wszystkie wyżej wymienione przypadki użycia powinny być zaimplementowane i przetestowane. W tym celu stworzono testy behawioralne przy użyciu narzędzia *cucumber*. Testy te przyjmują formę scenariuszy dla konkretnych przypadków użycia (listing 3.4).

1 Scenario: Registering a user account



Rys. 3.3: Szkic strony aukcji

```

2  Given I am on the home page
3  And no emails have been sent
4  When I follow "Sign_up"
5  And I fill in the following:
6      | user_email          | user@example.com |
7      | user_password       | monkey          |
8      | user_password_confirmation | monkey          |
9  And I press "Sign_up"
10 Then "user@example.com" should receive an email
11 When "user@example.com" opens the email
12 Then I should see "Confirmation_instructions" in the email subject
13 And I should see "confirm_your_account_through_the_link_below" in the email body
14 And I should see "Confirm_my_account" in the email body
15 When I follow "Confirm_my_account" in the email
16 And I should see "Sign_out"

```

Listing 3.4: Przykładowy scenariusz opisujący proces rejestracji nowego użytkownika

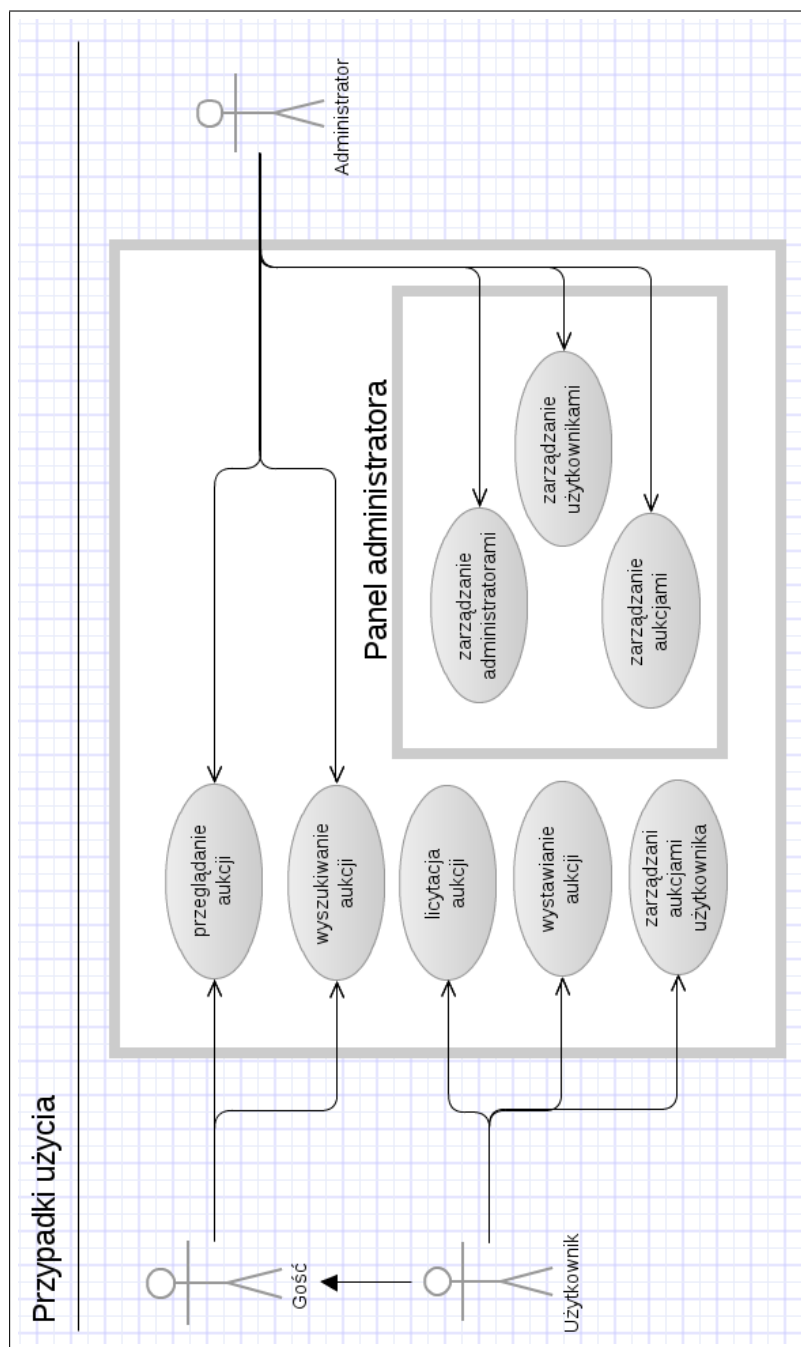
### 3.2.3 Zadania dotyczące implementacji i ich realizacja

Zadania, które należało wykonać, aby zaimplementować system *auctioner* są następujące (listing 3.5):

1. inicjacja środowiska – zainstalowanie niezbędnych narzędzi, konfiguracja środowiska oraz stworzenie nowego projektu Ruby on Rails,
2. konfiguracja projektu Ruby on Rails – „podpięcie” wtyczek (gem), ustawienia konfiguracyjne dla projektu, utworzenie repozytorium *Git* projektu, wykonanie czynności mających na celu przygotowanie projektu dla realizacji jego architektury i funkcjonalności,
3. przygotowanie aplikacji do obsługi wielu wersji językowych,
4. utworzenie modeli użytkowników i administratorów – utworzenie schematu tabeli w bazie danych, implementacja klas opakowujących, utworzenie prostych kontrolerów zarządzających modelami,
5. dodanie funkcjonalności rejestracji/logowania – podpięcie systemu autentykacji do istniejących modeli użytkownika i administratora, utworzenie widoków dla funkcjonalności rejestracji, logowania, przypomnienia hasła, aktywacji konta oraz przygotowanie wersji językowych dla funkcjonalności,
6. utworzenie modelu dla aukcji – przygotowanie schematu bazy danych, utworzenie zależności pomiędzy modelami aukcji i użytkownika, utworzenie kontrolerów zarządzających aukcjami, utworzenie widoków dla aukcji, utworzenie ścieżek wywołań,
7. dodanie możliwości wyszukiwania – implementacja systemu zapytań do bazy danych, obsługa tworzonych kwerend,
8. utworzenie systemu publikacji i podbijania aukcji – dodanie maszyny stanów dla modelu aukcji, implementacja kontrolerów zarządzających zmianami stanu, zmiany w widokach dla funkcjonalności.

1	TicId	Title	State	Date	Assgn	Tags
2	-----					
3	69418c	Create a new rails projec...	reso...	11/07	placek	environment
4	36ce13	Prepare L10n and I18n env...	reso...	11/07	placek	environment
5	fca367	Prepare a devise models f...	reso...	11/07	placek	users
6	ed753d	Create model for auction;	reso...	11/07	placek	
7	b2296e	Prepare a publication sys...	reso...	11/07	placek	auction,state

Listing 3.5: Lista zadań w aplikacji *ticgit*



Rys. 3.4: Diagram obrazujący przypadki użycia oraz aktorów stworzonego systemu

## 3.3 Projekt *auctioneer*

Poniżej zaprezentowano rezultat pracy nad projektem *auctioneer*.

### 3.3.1 Architektura

Aplikacje pisane przy użyciu aplikacji szkieletowej Ruby on Rails opierają się na architekturze *MVC* (ang. *model-view-controller*). Oznacza to, że w aplikacji takiej wyróżnione są warstwy odpowiedzialne za dostęp do danych i ich „obróbkę”, realizację logiki biznesowej aplikacji oraz prezentację (interfejs graficzny).

#### Warstwa danych – model

Ruby on Rails wykorzystuje moduł `ActiveRecord` do opakowania rekordów, pochodzących z relacyjnych baz danych, odpowiadającymi im obiektami. Dzięki temu programista może traktować tabele bazy danych jako klasę, a rekordy tej tabeli jako jej instancje.

admins

reprezentacja danych modelu administratora; zawiera adres e-mail, zaszyfrowane hasło, stempel czasu (utworzenie i edycja rekordu) oraz dane logowania (stempel czasu, adres IP, z którego się zalogowano),

users

reprezentacja danych modelu użytkownika; zawiera adres e-mail, zaszyfrowane hasło, stempel czasu, dane logowania oraz dane aktywacji konta,

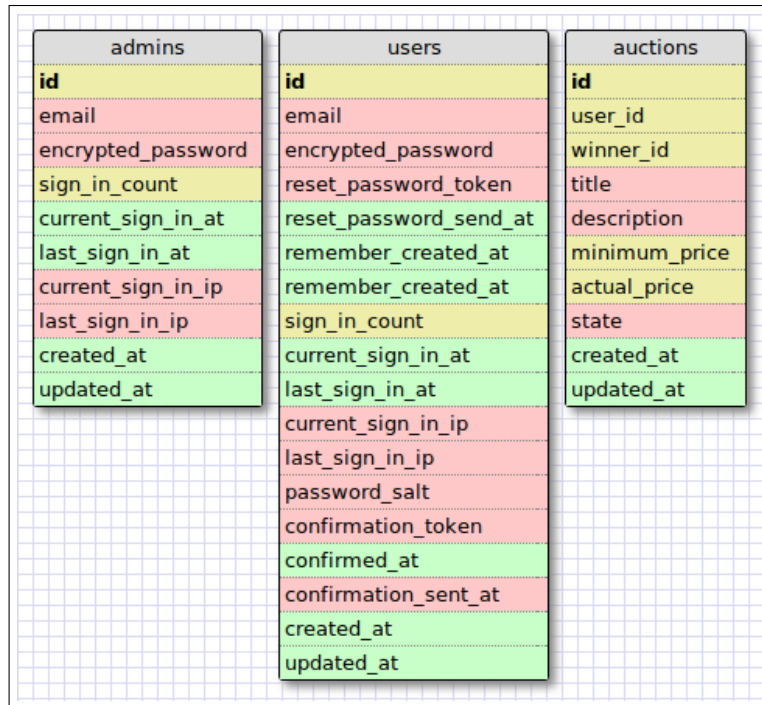
auctions

reprezentacja danych modelu aukcji; zawiera tytuł, opis, cenę minimalną i aktualną aukcji, aktualny stan, klucze (identyfikatory) użytkownika-właściciela i użytkownika-zwycięzcy oraz stempel czasu.

Na rysunku 3.5 zamieszczono schemat tabeli bazy danych projektu *auctioneer*. Listingi 3.6, 3.7, 3.8 pokazują klasy opakowujące dane dla modeli projektu *auctioneer*.

```
1 class Auction < ActiveRecord::Base
2   EXPIRATION_AFTER = 7.days
3
4   belongs_to :user
5   belongs_to :winner, class_name: 'User'
6   validates :title, presence: true
7   validates :minimum_price, presence: true
8
9   scope :created, where(state: :new)
10  scope :public, where(state: :public)
11  scope :closed, where(state: :closed)
12  scope :won, where(state: :won)
```





Rys. 3.5: Schemat bazy danych projektu *auctioneer*

```

13 scope :finished, where('state_="closed"_OR_state_="won"')
14 scope :won_by, ->(user) { where(state: :won).where(winner_id: user.id) }
15
16 def self.about_to_finish
17   time = Time.now
18   now = Time.new(time.year, time.month, time.day, time.hour, 0)
19   where(state: :public).
20   where('created_at_<?', now - EXPIRATION_AFTER + 1.hour).
21   where('created_at_>=?', now - EXPIRATION_AFTER)
22 end
23
24 # scope used in search engine
25 def self.search(query)
26   if query
27     phrase = "%#{query}%"
28     where('(title_LIKE_)_OR_(description_LIKE_)', phrase, phrase)
29   else
30     scoped
31   end
32 end
33
34 # closing every expired auction
35 def self.close_auctions

```

```

36     self.about_to_finish.each do |auction|
37         auction.close if auction.winner.nil?
38         auction.win if auction.winner.present?
39     end
40 end
41
42 state_machine initial: :new do
43     after_transition any => :public do |auction, transition|
44         auction.actual_price = auction.minimum_price
45         auction.save!
46     end
47
48     event :publish do
49         transition new: :public, if: ->(auction) {
50             auction.description.present?
51         }
52     end
53
54     event :close do
55         transition public: :closed
56     end
57
58     event :win do
59         transition public: :won, if: ->(auction) {
60             auction.actual_price != auction.minimum_price && auction.winner.present?
61         }
62     end
63
64     event :republish do
65         transition closed: :public
66     end
67 end
68
69 end

```

Listing 3.6: Klasa reprezentacji modelu aukcji

```

1 class Admin < ActiveRecord::Base
2     devise :database_authenticatable, :trackable, :validatable,
3           :timeoutable
4     attr_accessible :email, :password, :password_confirmation
5     validates_presence_of :email
6     validates_uniqueness_of :email, case_sensitive: false
7
8     scope :email_like, ->(email) { where('admins.email_like_', email) }
9 end

```

Listing 3.7: Klasa reprezentacji modelu administratora

```

1 class User < ActiveRecord::Base
2   devise :database_authenticatable, :registerable, :recoverable,
3         :rememberable, :trackable, :validatable, :confirmable
4   attr_accessible :email, :password, :password_confirmation,
5         :remember_me
6   validates_presence_of :email
7   validates_uniqueness_of :email, case_sensitive: false
8   has_many :auctions
9
10  scope :email_like, ->(email) { where('users.email_like?', email) }
11 end

```

Listing 3.8: Klasa reprezentacji modelu użytkownika

## Warstwa logiki – kontroler

*Kontroler* jest elementem decyzyjnym realizującym założenia logiki biznesowej projektu. W Ruby on Rails kontrolerami są pewne klasy, które grupują w sobie metody (tak zwane akcje). Akcje są wykonywane podczas każdego zapytania HTTP. O tym, która akcja zostanie wykonana, decyduje system rozpoznawania ścieżek i parametrów zapytania – routes. Listing 3.9 prezentuje przykładowy kontroler dla aplikacji *auctioneer*.

```

1 class Admin::UsersController < ApplicationController
2   before_filter :authenticate_admin!
3   layout 'admin'
4
5   def index
6     @users = User.email_like(params[:like] || '%').paginate(page: params[:page])
7   end
8
9   def destroy
10    @user = User.find(params[:id]).destroy
11    flash[:notice] = t('admin.registrations.destroy')
12    redirect_to admin_users_path
13  end
14
15  def login
16    sign_in(:user, User.find(params[:id]))
17    redirect_to root_path
18  end
19
20  def confirm
21    User.find(params[:id]).confirm!
22    redirect_to admin_users_path
23  end

```

24 **end**

Listing 3.9: Przykładowy kontroler `Admin::UsersController` odpowiadający za akcje związane z zarządzaniem użytkownikami w panelu administratora

## Warstwa prezentacji – widok

Widoki to szablony, generujące rezultaty – odpowiedzi na zadane zapytania protokołu HTTP. Są one wynikiem działania akcji kontrolera. W aplikacji *auctioneer* użyto metajęzyka `haml` do tworzenia takich szablonów. Listing 3.10 przedstawia przykładowy szablon widoku.

```
1 = form_tag admin_users_path, method: :get do
2   = label_tag :email, "Filter_by_email:"
3   = text_field_tag :like, h(params[:email])
4   = submit_tag "Filter"
5   .grey
6     use '%' to match anything (like '%examp%')
7
8 - unless params[:like].nil?
9   %p
10     Filtered with
11     %strong
12       = h(params[:like])
13
14 = will_paginate(@users)
15
16 %table
17   %tr
18     %th Email
19     %th Created at
20     %th Confirmed at
21     %th Last signed in
22     %th Sign in IP
23     %th{ colspan: 2 } Actions
24 - @users.each do |user|
25   %tr
26     %td= user.email
27     %td= user.created_at
28     %td= user.confirmed_at.present? ? user.confirmed_at : 'not_confirmed_yet'
29     %td= user.last_sign_in_at.present? ? user.last_sign_in_at : 'not_signed_in_yet'
30     %td= user.last_sign_in_at.present? ? user.last_sign_in_ip : 'not_signed_in_yet'
31
32     - if user.confirmed?
33       %td= link_to 'Sign_in', admin_user_login_path(user)
34     - else
35       %td= link_to 'Confirm', admin_user_confirm_path(user)
36       %td= link_to 'Delete', admin_user_delete_path(user), confirm: 'Are_you_sure?'
```

```
37
38 = will_paginate(@users)
```

Listing 3.10: Przykładowy widok – lista użytkowników w panelu administratora

### 3.3.2 Testy

W realizacji poszczególnych funkcjonalności systemu ważną rolę spełniają testy. Stanowią one potwierdzenie poprawnego działania poszczególnych komponentów, a także chronią programistę przed popełnieniem nie przewidzianego błędu.

#### Testy jednostkowe

Testy jednostkowe `rspec` sprawdzają poprawność działania elementów logiki biznesowej, metod pomocniczych dla warstwy danych i widoku oraz reakcji na zapytania HTTP. Testy jednostkowe znajdują się w katalogu `spec`. Po wywołaniu komendy `rake spec` aplikacja zostaje przetestowana. Wyniki takiego testowania zawiera listing 3.11.

```
1 $ rake spec
2 ApplicationController
3   routing
4     routes to static_admin/admins#dashboard
5     routes to static_admin/admins#index
6     routes to static_admin/users#index
7     routes to static_static_pages#landing
8
9 AuctionsController
10  routing
11    routes to #index
12    routes to #show
13    routes to #new
14    routes to #edit
15    routes to #create
16    routes to #update
17    routes to #destroy
18
19 Auction
20   about_to_finish
21     should not be empty
22   about_to_finish
23     should be empty
24   about_to_finish
25     should be empty
26   close_auctions
27     should eq 1
28
```

```

29 AdminHelper
30   total_count_of
31     should return a quantity of all records in model
32
33 AuctionsHelper
34   short_description
35     should return a shorten text with no html tags and ended with '...'
36
37 ApplicationHelper
38   after_sign_in_path_for
39     should return a proper url
40
41 Finished in 6.98 seconds
42 18 examples, 0 failures

```

Listing 3.11: Wyniki testowania aplikacji narzędziem *rspec*

## Testy behawioralne

Testy behawioralne sprawdzają zachowanie aplikacji przy zadanych warunkach. W projektach Ruby on Rails zwykle używa się w tym celu narzędzia *cucumber*. Testy takie składają się z tak zwanych scenariuszy. Scenariusz to lista kroków, w których realizowane są poszczególne działania w aplikacji oraz sprawdzane założenia. Testy behawioralne wraz z konfiguracją środowiska testującego znajdują się w katalogu *features*. Po wywołaniu komendy `rake cucumber` aplikacja zostaje przetestowana. Wyniki takiego testowania zawiera listing 3.12.

```

1 Feature: User accounts
2   In order to have a private account
3     A user
4       Wants to manage it
5
6   Scenario: Registering a user account
7     Given I am on the home page
8       And no emails have been sent
9       When I follow "Sign_up"
10        And I fill in the following:
11          | user_email          | user@example.com |
12          | user_password       | monkey           |
13          | user_password_confirmation | monkey           |
14        And I press "Sign_up"
15        Then "user@example.com" should receive an email
16        When "user@example.com" opens the email
17          Then I should see "Confirmation_instructions" in the email subject
18          And I should see "You_can_confirm_your_account" in the email body
19          And I should see "Confirm_my_account" in the email body
20        When I follow "Confirm_my_account" in the email

```

```

21     And I should see "Sign_out"
22
23 Scenario: Changing users password
24     Given a user "quentin"
25     And I am on the home page
26     When I follow "Sign_in"
27     And I follow "Forgot_your_password?"
28     And I fill in the following:
29         | user_email | quentin@example.com |
30     And I press "Send_me_reset_password_instructions"
31     Then I should see "You_will_receive_an_email_with_instructions"
32     And "quentin@example.com" should receive an email
33     When "quentin@example.com" opens the email
34     Then I should see "Reset_password_instructions" in the email subject
35     And I should see "link_to_change_your_password" in the email body
36     When I follow "Change_my_password" in the email
37     Then I should see "Change_your_password"
38     When I fill in the following:
39         | user_password | monkey |
40         | user_password_confirmation | monkey |
41     And I press "Change_my_password"
42
43 Scenario: Resend a confirmation instructions
44     Given I am on the home page
45     And an unconfirmed user "user"
46     And a clear email queue
47     When I follow "Sign_up"
48     And I follow "Didn't_receive_confirmation_instructions?"
49     Then I should see "Resend_confirmation_instructions"
50     When I fill in the following:
51         | user_email | user@example.com |
52     And I press "Resend_confirmation_instructions"
53     Then I should see "You_will_receive_an_email_with_instructions"
54     And "user@example.com" should receive an email
55     When "user@example.com" opens the email
56     Then I should see "Confirmation_instructions" in the email subject
57     And I should see "You_can_confirm_your_account" in the email body
58
59 Scenario: Logging in and out
60     Given a user "quentin"
61     And I am on the home page
62     When I follow "Sign_in"
63     And I fill in the following:
64         | user_email | quentin@example.com |
65         | user_password | secret |
66     And I press "Sign_in"

```

Listing 3.12: Przykładowy scenariusz opisujący system rejestracji i logowania użytkowników

### 3.3.3 Dziennik realizacji zadań

Realizacja poszczególnych zadań ma swoje odzwierciedlenie w historii kodu w repozytorium *Git*. Poniżej przedstawiono historię zmian kodu.

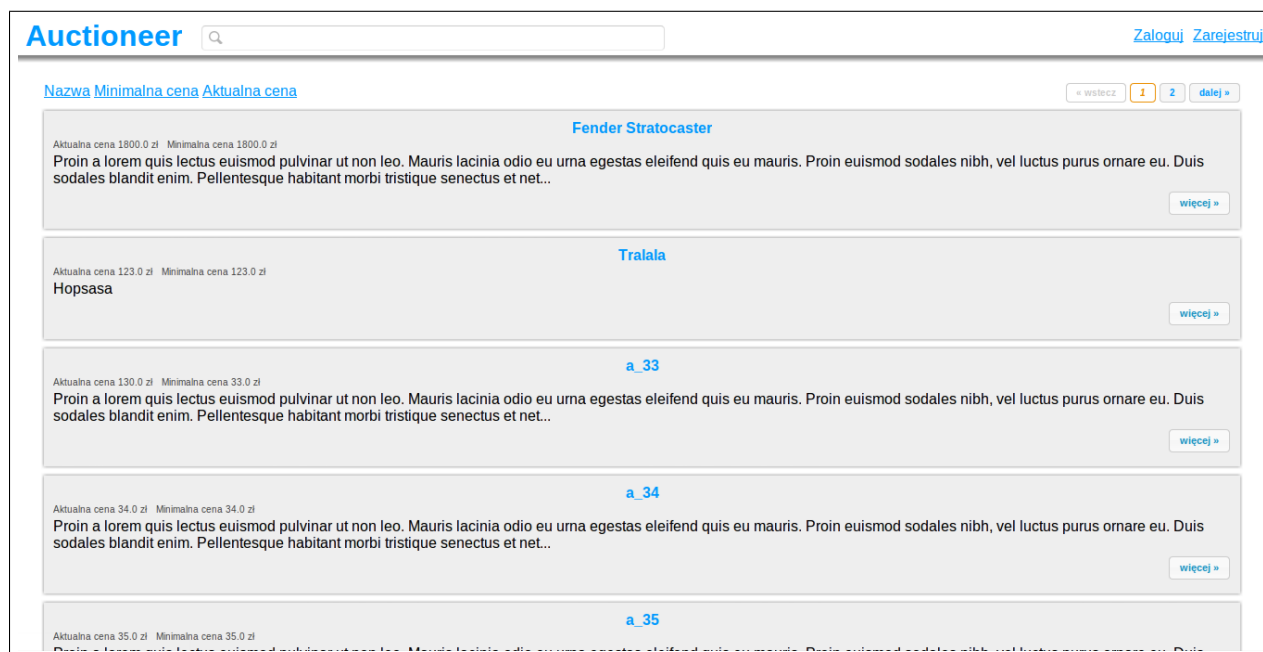
```
* 5f69148: changes in report; closed chapter describing application
* 1b8d6b1: changes in report; minor changes in application
* cf54641: removed ruby-debug from Gemfile
* 8c2da87: another changes due to report
* 99ba8ff: added new content to report; some small corrections
* d360dfa: fixed spec tests for auction model
* fd1c522: changes in report; added new chapter
* a2fb439: [#134409] added simple relancing system with hourly checking for winners
* 9eb16d9: [#8269df] ordering the search results by title, minimum and actual price
* 4707b6a: some more resolution to report
* f73219b: changes in report due to suggestions
* 9bd648b: [#378254] simple search engine added
* c5a3008: [#b2296e] auctions are treated as state machine
* c9526a7: Merging branches
|\
| * 2c1e199: [#fca367 and #ed753d] simple schema of auctioning system
| * bcc6b5a: [#fca367] changes in admin and user models; prepared devise system
* | b234283: removed pdf file from documentation
* | 6ce2ac8: more specific documentation about used tools
* | 2972c55: some logical changes in report
* | 2017f51: added a doc directory containing a documentation
* | 8c8cb9a: [#fca367] added admin and user models;
|/
* 836ffa3: removed unnecessary README_FOR_APP file
* 17550e4: [#36ce13] added localization scopes
* f42d5dc: first commit; new project created; initialized gems
```

## 3.4 Podręcznik użytkownika

Poniżej zamieszczono opis stworzonej aplikacji od strony osoby jej użytkującej. Osobą taką może być jeden z trzech aktorów: gość, zwykły użytkownik oraz administrator. Możliwe działania dla poszczególnych



aktorów tworzą scenariusze dla poszczególnych przypadków użycia. Założono, że punktem wyjścia jest odwiedzenie przez aktora strony głównej projektu (Rys. 3.6).



Rys. 3.6: Strona główna serwisu *auctioneer* – lista wystawionych aukcji

### 3.4.1 Opis projektu od strony gościa

**Strona główna** Na stronie głównej znajdują się następujące aktywne elementy:

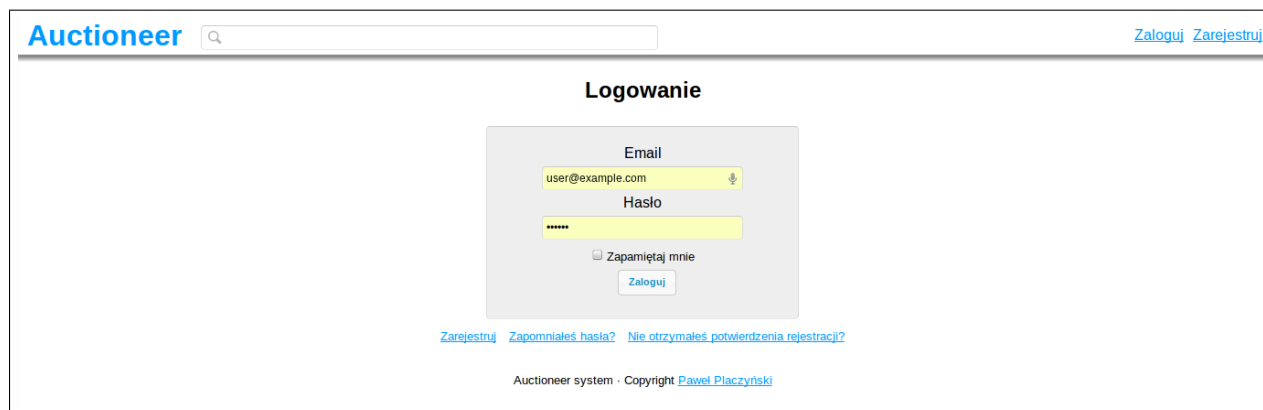
- pole tekstowe wyszukiwarki aukcji,
- link „Zaloguj”,
- link „Zarejestruj”,
- link „Nazwa”,
- link „Minimalna cena”,
- link „Aktualna cena”,
- zestaw przycisków: „wstecz”, „1”, „2”, ..., „dalej”,
- lista aukcji,
- link-nagłówek „Auctioneer”.

Pole tekstowe wyszukiwarki aukcji pozwala na wyselekcjonowanie aukcji, znajdujących się w liście poniżej, według zadanych kryteriów. Wpisanie wyszukiwanej frazy oraz wciśnięcie klawisza **Enter** spowoduje przejście do strony z wynikami wyszukiwania (Rys. 3.7). Strona z wynikami wyszukiwania jest podobna do strony głównej, z tą różnicą, że ponad wynikami wyszukiwania widoczna jest informacja na temat wyszukiwanej frazy.



Rys. 3.7: Działanie wyszukiwarki aukcji serwisu *auctioneer*

Link „Zaloguj” przekierowuje gościa do formularza logowania (Rys. 3.8). Formularz logowania opisany został poniżej. Link „Zarejestruj” przekierowuje do formularza rejestracji (Rys. 3.9). Zestaw odsyłaaczy: „Nazwa”, „Minimalna cena” oraz „Aktualna cena” zmienia kolejność wyświetlania aukcji, sortując je według nazwy, minimalnej oraz aktualnej ceny (ponowne wybranie linku odwraca kolejność sortowania). Link-nagłówek „Auctioneer” prowadzi zawsze na stronę główną.



Rys. 3.8: Ekran logowania

Zestaw przycisków: „Wstecz”, „1”, „2”, ..., „Dalej” pozwala przeglądać kolejne strony wystawionych aukcji (mechanizm „paginacji stron”).

**Aukcje** Na stronie głównej znajduje się lista aukcji. Każda aukcja ma przy swoim opisie przycisk „Więcej” prowadzący do strony aukcji (Rys. 3.10 oraz Rys. 3.11).

Auctioneer

Zaloguj Zarejestruj

### Rejestracja

Email  
user@example.com

Hasło  
\*\*\*\*\*

Potwierdź hasło

Zarejestruj

[Zaloguj](#) [Zapomniałeś hasła?](#) [Nie otrzymałeś potwierdzenia rejestracji?](#)

Auctioneer system - Copyright Paweł Piaczyński

Rys. 3.9: Ekran rejestracji

Strona aukcji zawiera pełny opis wystawionej aukcji. Przycisk „Wróć” pozwala wrócić na stronę główną, a przycisk „Podbij” prowadzi do formularza podbijania aukcji (Rys. 3.12). Jeśli użytkownik nie jest zalogowany to po naciśnięciu przycisku „Podbij” zostaje on przekierowany do formularza logowania. Po prawidłowym zalogowaniu, użytkownik zostaje automatycznie przekierowany do formularza podbijania aukcji.

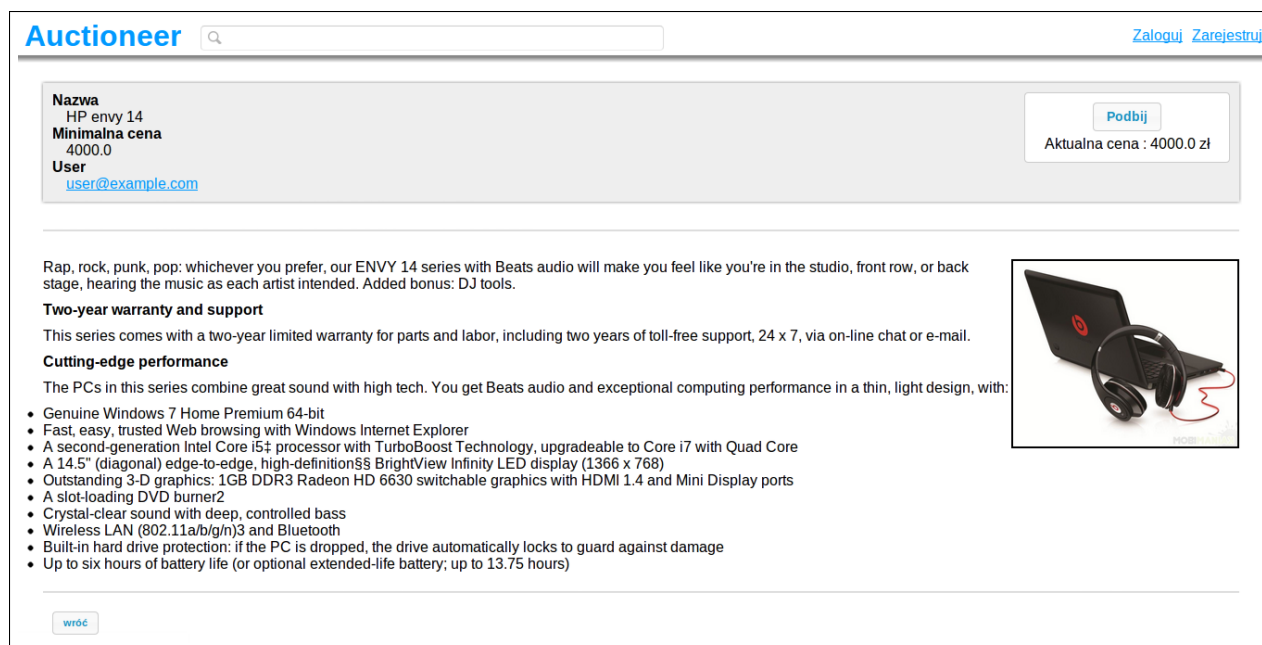
**Formularz logowania** Formularz logowania umożliwia niezalogowanemu użytkownikowi otworzyć nową, bezpieczną sesję. Pola tekstowe pozwalają na wprowadzenie danych logowania: adresu e-mail oraz hasła. Użycie opcji „Zapamiętaj mnie” skutkuje tym, że sesja użytkownika nie wygaśnie, gdy użytkownik będzie nieaktywny przez dłuższy czas. Przycisk „Zaloguj” służy do zatwierdzenia operacji logowania.

Jeżeli operacja logowania przeszła pomyślnie (użytkownik o podanym adresie e-mail oraz hasle posiada konto w serwisie oraz zostało aktywowane, ponadto adres e-mail oraz hasło zostały poprawnie podane), to otwarta zostaje sesja użytkownika oraz zostanie on przekierowany do panelu użytkownika. Akcje dla zalogowanego użytkownika opisuje podrozdział 3.4.2.

Pod formularzem znajdują się odnośniki pomagające użytkownikowi zarejestrować się, bądź odzyskać utracone hasło. Link „Zarejestruj” przekierowuje do formularza rejestracji, link „Zapomniałeś hasła?” przekierowuje do formularza zmiany hasła (Rys. 3.13), natomiast link „Nie otrzymałeś potwierdzenia rejestracji?” przekierowuje do formularza ponownego wysłania potwierdzenia rejestracji, gdy takie potwierdzenie nie zostało poprawnie wysłane na adres e-mail użytkownika (Rys. 3.14).

### 3.4.2 Opis projektu od strony użytkownika

**Panel użytkownika (Kokpit)** Po zalogowaniu, użytkownik zostaje przekierowany do panelu użytkownika (Rys. 3.15). W „nagłówku” strony znajduje się zestaw odnośników: link „Kokpit” prowadzi do panelu użytkownika, link „Profil” prowadzi do formularza edycji profilu (Rys. 3.16), a link „Wyloguj” zamyka sesję użytkownika i przekierowuje na stronę główną.



Rys. 3.10: Strona aukcji zawierająca detale. Widok dla gościa lub zalogowanego użytkownika nie wystawiającego aukcję

Na panelu użytkownika widoczne są zakładki: *Nowe*, *Wystawione*, *Zakończone*, *Wygrane*. Otwierają one kolejno: listę nowo utworzonych aukcji użytkownika, listę aukcji aktualnie wystawionych przez użytkownika, listę aukcji zakończonych oraz listę aukcji wygranych przez użytkownika.

Każda aukcja, z dowolnej, wyżej wymienionej listy, posiada dwa przyciski: „edytuj” – prowadzący do formularza edycji aukcji (Rys. 3.17) oraz „usuń” – powodujący usunięcie aukcji po uprzednim potwierdzeniu operacji. Ponadto aukcja posiada przyciski zmiany stanu: aukcje z listy *Nowe* posiadają przycisk „wystaw”, aukcje z listy *Wystawione* – „zakończ” oraz aukcje z listy *Zakończone* – „wystaw ponownie”.

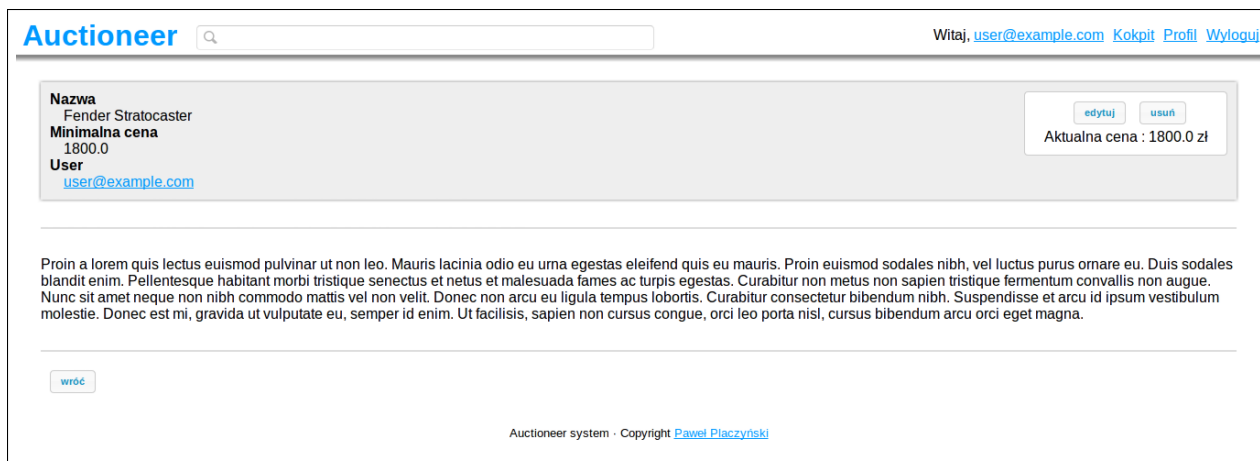
**Panel edycji aukcji** Panel edycji aukcji posiada pole tekstowe zmiany nazwy (tytułu) aukcji, pole tekstowe zmiany ceny minimalnej aukcji oraz pole tekstowe zaopatrzone w zestaw narzędzi, służących do formatowania tekstu, do edycji opisu aukcji. Przycisk „Aktualizuj” zatwierdza zmiany.

### 3.4.3 Opis projektu od strony administratora

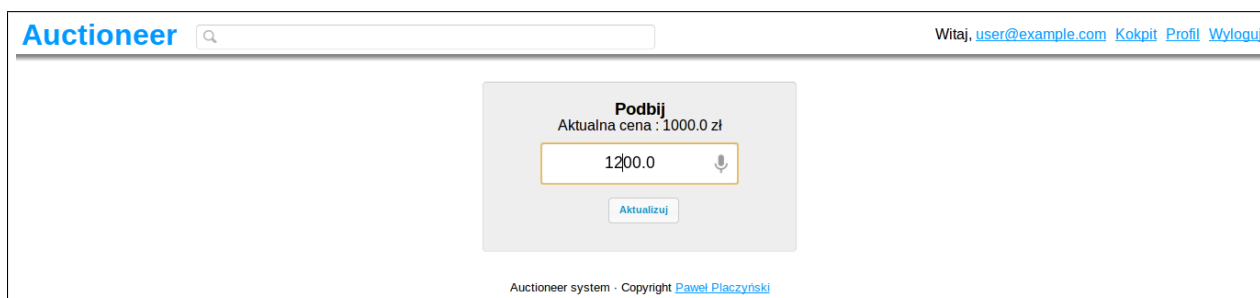
Aby odwiedzić panel administratora (Rys. 3.18) należy podać ścieżkę *adres\_bazowy\_aplikacji/admin*. Wyświetli się wówczas formularz logowania administratora. Podając odpowiednie dane (adres e-mail oraz hasło) można zarządzać aplikacją z poziomu administratora.

W panelu administratora znajdują się:

- Link „Leave panel”, który pozwala wyjść z panelu, pozostając zalogowanym jako administrator. Link ten przekierowuje na stronę główną.



Rys. 3.11: Strona aukcji zawierająca detale. Widok dla zalogowanego użytkownika wystawiającego aukcję



Rys. 3.12: Formularz podbijania aukcji

- Link „Sign out” umożliwia wylogowanie administratora.
- Link „Admins” pokazuje listę administratorów serwisu. Lista ta udostępnia następujące akcje: wyszukanie administratora według adresu e-mail (pole tekstowe oznaczone etykietą „Filter”) oraz usunięcie administratora (link „Delete”, przy czym nie jest możliwe usunięcie aktualnie zalogowanego administratora).
- Link „Users” pokazuje listę użytkowników. Lista ta pozwala na wyszukanie użytkownika według adresu e-mail (pole tekstowe oznaczone etykietą „Filter”), usunięcie użytkownika (link „Delete”) oraz zalogowanie się jako użytkownik (link „Login”).
- Link „Auctions” pokazuje listę aukcji. Lista pozwala na wyszukanie aukcji według tytułu lub opisu aukcji (pole tekstowe „Filter”) oraz usunięcie aukcji (link „Delete”).

**Auctioneer**

[Zaloguj](#) [Zarejestruj](#)

### Przypomnienie hasła

Email

[Wyslij instrukcje](#)

[Zaloguj](#) [Zarejestruj](#) [Nie otrzymałeś potwierdzenia rejestracji?](#)

Auctioneer system - Copyright [Paweł Placzyński](#)

Rys. 3.13: Formularz przypomnienia hasła

**Auctioneer**

[Zaloguj](#) [Zarejestruj](#)

### Ponowne wysłanie potwierdzenia rejestracji

Email

[Wyslij instrukcje](#)

[Zaloguj](#) [Zarejestruj](#) [Zapomniałeś hasła?](#)

Auctioneer system - Copyright [Paweł Placzyński](#)

Rys. 3.14: Formularz ponownego wysłania potwierdzenia rejestracji

**Auctioneer**

Witaj, [user@example.com](#) [Kokpit](#) [Profil](#) [Wyloguj](#)

**Nowe** **Wystawione** **Zakończone**

**Fender Stratocaster**

Aktualna cena 1800.0 zł Minimalna cena 1800.0 zł

Proin a lorem quis lectus euismod pulvinar ut non leo. Mauris lacinia odio eu urna egestas eleifend quis eu mauris. Proin euismod sodales nibh, vel luctus purus ornare eu. Duis sodales blandit enim. Pellentesque habitant morbi tristique senectus et net...

[zakończ](#) [edytuj](#) [usuń](#) [więcej »](#)

**Tralala**

Aktualna cena 123.0 zł Minimalna cena 123.0 zł

Hopsasa

[zakończ](#) [edytuj](#) [usuń](#) [więcej »](#)

**tralala**

Aktualna cena 1234.0 zł Minimalna cena 1234.0 zł

No co ty?

[zakończ](#) [edytuj](#) [usuń](#) [więcej »](#)

Auctioneer system - Copyright [Paweł Placzyński](#)

Rys. 3.15: Kokpit użytkownika wraz z listami aukcji użytkownika



## Admins panel

[Admins dashboard](#) [Leave panel](#) [Sign out](#) [Admins \(1\)](#) [Users \(2\)](#) [Auctions \(44\)](#) Filter:   use %' to match anything (like %sexamp%)

« previous 1 2 next »

Title	State	Minimum price	Actual price	Created at	Actions
MSI Wind u100	closed	1200.0	1200.0	2011-11-20 17:59:54 UTC	<a href="#">Delete</a>
Fender Stratocaster	public	1800.0	1800.0	2011-11-20 21:17:26 UTC	<a href="#">Delete</a>
a_1	new	1.0		2011-11-20 22:31:37 UTC	<a href="#">Delete</a>
a_2	new	2.0		2011-11-20 22:31:37 UTC	<a href="#">Delete</a>
a_4	new	4.0		2011-11-20 22:31:37 UTC	<a href="#">Delete</a>
a_5	new	5.0		2011-11-20 22:31:37 UTC	<a href="#">Delete</a>
a_6	new	6.0		2011-11-20 22:31:38 UTC	<a href="#">Delete</a>
a_7	new	7.0		2011-11-20 22:31:38 UTC	<a href="#">Delete</a>
a_8	new	8.0		2011-11-20 22:31:38 UTC	<a href="#">Delete</a>
a_9	new	9.0		2011-11-20 22:31:38 UTC	<a href="#">Delete</a>
a_10	new	10.0		2011-11-20 22:31:38 UTC	<a href="#">Delete</a>
a_11	new	11.0		2011-11-20 22:31:38 UTC	<a href="#">Delete</a>
a_12	new	12.0		2011-11-20 22:31:39 UTC	<a href="#">Delete</a>
a_13	new	13.0		2011-11-20 22:31:39 UTC	<a href="#">Delete</a>
a_14	new	14.0		2011-11-20 22:31:39 UTC	<a href="#">Delete</a>
a_15	new	15.0		2011-11-20 22:31:39 UTC	<a href="#">Delete</a>
a_16	new	16.0		2011-11-20 22:31:39 UTC	<a href="#">Delete</a>
a_17	new	17.0		2011-11-20 22:31:39 UTC	<a href="#">Delete</a>
a_18	new	18.0		2011-11-20 22:31:40 UTC	<a href="#">Delete</a>
a_19	new	19.0		2011-11-20 22:31:40 UTC	<a href="#">Delete</a>
a_20	new	20.0		2011-11-20 22:32:15 UTC	<a href="#">Delete</a>
a_21	new	21.0		2011-11-20 22:32:15 UTC	<a href="#">Delete</a>
a_22	new	22.0		2011-11-20 22:32:15 UTC	<a href="#">Delete</a>

Rys. 3.18: Panel administratora z listą aukcji



## **Rozdział 4**

# **Podsumowanie**

Aplikacja szkieletowa Ruby on Rails ma wiele zalet pozwalających na pracę przy pomocy metodyki programowania zwinnego. Struktura aplikacji opartych na frameworku Ruby on Rails jest przejrzysta, jasna i zorganizowana. Kod takiej aplikacji jest czytelny, a do jego zrozumienia wystarczy znajomość języka angielskiego.

Ponadto ściśle wyodrębnione warstwy aplikacji pozwalają na logiczny podział zadań poszczególnych komponentów oraz na uniknięcie nadmiarowych bądź nieoczekiwanych operacji. Zapewnia to bezpieczeństwo oraz dobrą kontrolę nad realizacją funkcjonalności.

Ruby on Rails posiada szereg wtyczek, dodatków, które realizują mniej lub bardziej skomplikowane działania. Można nimi w prosty sposób zarządzać, co sprawia, że praca nad aplikacją jest prostsza i bardziej efektywna.

## 4.1 Obserwacje

Podczas realizacji pracy poczyniono następujące obserwacje:

- Język *Ruby* ze względu na swoją naturę umożliwia swobodne tworzenie dowolnego typu aplikacji. Możliwość wyboru pomiędzy programowaniem strukturalnym, obiektowym, funkcyjnym lub nawet metaprogramowaniem nie ogranicza programisty, a wręcz pozwala mu na szybszą implementację założeń projektowych bez potrzeby zastanawiania się nad szczegółami.
- System zarządzania wtyczkami *RubyGems* wraz z pomocą narzędzia *bundler* zapewnia doskonałą kontrolę nad zależnościami projektu bez potrzeby troski o instalację wtyczek.
- Ścisły podział aplikacji Ruby on Rails na warstwy (model MVC) pozwala określić dokładnie zadania i cele poszczególnych komponentów. Umożliwia to lepszą kontrolę nad realizacją funkcjonalności oraz eliminację błędów mogących prowadzić do poważnych usterek bezpieczeństwa danych biznesowych.
- Możliwość hierarchizacji komponentów warstwy logiki (kontrolerów) pomaga w dobrej organizacji zakresu działań i akcji przeprowadzanych przez użytkowników aplikacji na danych.
- Podział warstwowy pozwala także zwiększyć czytelność kodu, co znacznie przyspiesza i usprawnia pracę nad nim.
- Interesujący system tworzenia kwerend modułu *ActiveRecord* pozwala na szybkie i czytelne konstruowanie zapytań bazy danych bez potrzeby konstruowania skomplikowanych kwerend w języku *SQL*. Pozwala to także na wykonywanie operacji na bazie danych niezależnie od systemu zarządzania bazą danych (DBMS).
- Testy jednostkowe i behawioralne pozwalają szybko sprawdzić poprawność działania poszczególnych elementów aplikacji bez potrzeby jej uruchamiania, czy wdrażania.

- Narzędzie cucumber wspomaga współpracę z potencjalnym klientem: scenariusze testujące funkcjonalność mogą być pisane przez programistę i przez klienta. W zasadzie tego typu scenariusze wraz z szkicami (mockupami) stanowią pełną dokumentację przypadków użycia (a zatem funkcjonalności, założeń projektu).
- Symulacja pracy w metodyce Scrum wyłania jasno określone cele, przez co zadania są szczegółowe oraz brak w nich niedopowiedzeń.
- System kontroli wersji pozwala na sprawną pracę nad projektem w przypadkach częstych zmian w projekcie. Jest on dobrym narzędziem, wspomagającym pracę w metodyce programowania zwinnego.

# Spis rysunków

1.1	Statystyki Google na temat aplikacji szkieletowych . . . . .	8
1.2	Program <i>tig</i> podczas pracy nad repozytorium projektu. . . . .	11
1.3	Webowy interfejs <i>ticgitweb</i> dla narzędzia zarządzania zadaniami <i>ticgit</i> . . . . .	12
1.4	Program <i>Vim</i> podczas pracy . . . . .	13
2.1	Schemat pracy w metodyce <i>Scrum</i> . . . . .	17
2.2	Narzędzie <i>PivotalTracker</i> . . . . .	19
3.1	Szkic panelu użytkownika . . . . .	35
3.2	Szkic wyszukiwarki aukcji . . . . .	36
3.3	Szkic strony aukcji . . . . .	37
3.4	Diagram obrazujący przypadki użycia oraz aktorów stworzonego systemu . . . . .	39
3.5	Schemat bazy danych projektu <i>auctioneer</i> . . . . .	41
3.6	Strona główna serwisu <i>auctioneer</i> – lista wystawionych aukcji . . . . .	49
3.7	Działanie wyszukiwarki aukcji serwisu <i>auctioneer</i> . . . . .	50
3.8	Ekran logowania . . . . .	50
3.9	Ekran rejestracji . . . . .	51
3.10	Strona aukcji zawierająca detale. Widok dla gościa lub zalogowanego użytkownika nie wystawiającego aukcję . . . . .	52
3.11	Strona aukcji zawierająca detale. Widok dla zalogowanego użytkownika wystawiającego aukcję . . . . .	53
3.12	Formularz podbijania aukcji . . . . .	53
3.13	Formularz przypomnienia hasła . . . . .	54
3.14	Formularz ponownego wysłania potwierdzenia rejestracji . . . . .	54
3.15	Kokpit użytkownika wraz z listami aukcji użytkownika . . . . .	54
3.16	Panel edycji profilu użytkownika . . . . .	55
3.17	Panel edycji aukcji . . . . .	55
3.18	Panel administratora z listą aukcji . . . . .	56

# Spis Listingów

2.1	Przykład prostej składni języka Ruby – algorytm DFS . . . . .	20
2.2	Algorytm DFS z zastosowaniem braku konwencji formatu . . . . .	21
2.3	Komentarze w języku Ruby . . . . .	24
2.4	Test <i>RSpec</i> testujący klasę <i>Array</i> . . . . .	25
2.5	Efekt uruchomienia testu jednostkowego z przykładu 2.4 . . . . .	26
2.6	Efekt uruchomienia testu jednostkowego z opcją <code>--format d</code> . . . . .	26
3.1	Zawartość katalogu <i>auctioneer</i> . . . . .	30
3.2	Plik <i>Gemfile</i> projektu <i>auctioneer</i> . . . . .	31
3.3	Plik <i>.gitignore</i> projektu <i>auctioneer</i> . . . . .	33
3.4	Przykładowy scenariusz opisujący proces rejestracji nowego użytkownika . . . . .	36
3.5	Lista zadań w aplikacji <i>ticgit</i> . . . . .	38
3.6	Klasa reprezentacji modelu aukcji . . . . .	40
3.7	Klasa reprezentacji modelu administratora . . . . .	42
3.8	Klasa reprezentacji modelu użytkownika . . . . .	43
3.9	Przykładowy kontroler <i>Admin::UsersController</i> odpowiadający za akcje związane z zarządzaniem użytkownikami w panelu administratora . . . . .	43
3.10	Przykładowy widok – lista użytkowników w panelu administratora . . . . .	44
3.11	Wyniki testowania aplikacji narzędziem <i>rspec</i> . . . . .	45
3.12	Przykładowy scenariusz opisujący system rejestracji i logowania użytkowników . . . . .	46

# Bibliografia

- [1] *Strona domowa projektu Ruby on Rails* <http://rubyonrails.pl> (stan na dzień 16 lutego 2012)
- [2] *Narzędzie Rspec do testów jednostkowych i funkcjonalnych* <http://rspec.info/> (stan na dzień 16 lutego 2012)
- [3] *Narzędzie Cucumber pozwalające na wykonywanie testów behavioralnych* <http://cukes.info> (stan na dzień 16 lutego 2012)
- [4] *Meta-język szablonów dokumentów XHTML* <http://haml-lang.com/> (stan na dzień 16 lutego 2012)
- [5] *Meta-język szablonów dokumentów CSS* <http://sass-lang.com/> (stan na dzień 16 lutego 2012)
- [6] *W pełni konfigurowalny system autentyfikacji użytkowników dla Ruby on Rails* <https://github.com/plataformatec/devise> (stan na dzień 16 lutego 2012)
- [7] *Plugin paginacji stron dla Ruby on Rails* [https://github.com/mislav/will\\_paginate](https://github.com/mislav/will_paginate) (stan na dzień 16 lutego 2012)
- [8] *Rozwinięty edytor HTML dla stron internetowych* <http://tinymce.moxiecode.com/> (stan na dzień 16 lutego 2012)
- [9] *Lekka i szybka relacyjna baza danych SQL* <http://www.sqlite.org/> (stan na dzień 16 lutego 2012)
- [10] *Biblioteka programistyczna języka JavaScript* [http://docs.jquery.com/Main\\_Page](http://docs.jquery.com/Main_Page) (stan na dzień 16 lutego 2012)
- [11] *System kontroli wersji Git* <http://git-scm.com> (stan na dzień 16 lutego 2012)
- [12] *ticgit – manager zadań projektowych dla projektów używających system kontroli wersji Git* <https://github.com/schacon/ticgit/wiki/> (stan na dzień 16 lutego 2012)
- [13] *Webowy interfejs dla narzędzia ticgit – ticgitweb* <https://github.com/schacon/ticgit/wiki/TicGitWeb> (stan na dzień 16 lutego 2012)
- [14] *Powłoka ZSH* <http://www.zsh.org/> (stan na dzień 16 lutego 2012)

- [15] *GNU screen* <http://www.gnu.org/software/screen/> (stan na dzień 16 lutego 2012)
- [16] *Rozbudowany edytor tekstu Vim* <http://www.vim.org/> (stan na dzień 16 lutego 2012)
- [17] *System kontroli wersji języka Ruby RVM* <https://rvm.beginrescueend.com/> (stan na dzień 16 lutego 2012)
- [18] *Sqliteman – graficzne narzędzie do zarządzania bazą danych Sqlite* <http://sqliteman.com/> (stan na dzień 16 lutego 2012)
- [19] *Heroku – narzędzie do wdrażania aplikacji webowych* <http://www.heroku.com/> (stan na dzień 16 lutego 2012)
- [20] *The Scrum Framework in 30 seconds*, [www.scrumalliance.org/pages/what\\_is\\_scrum](http://www.scrumalliance.org/pages/what_is_scrum) (stan na dzień 16 lutego 2012)
- [21] *emphAgile Manifesto*, [agilemanifesto.org/principles.html](http://agilemanifesto.org/principles.html) (stan na dzień 16 lutego 2012)
- [22] *Agile and Scrum programming*, [www.agileprogramming.org](http://www.agileprogramming.org) (stan na dzień 16 lutego 2012)
- [23] *Programowanie zwinne*, [http://pl.wikipedia.org/wiki/Programowanie\\_zwinne](http://pl.wikipedia.org/wiki/Programowanie_zwinne) (stan na dzień 16 lutego 2012)
- [24] *Konwencja języka Ruby*, <https://github.com/chneukirchen/styleguide/blob/master/RUBY-STYLE> (stan na dzień 16 lutego 2012)
- [25] *Dokumentacja standardu HTML5*, <http://dev.w3.org/html5/spec/Overview.html> (stan na dzień 16 lutego 2012)
- [26] *Dokumentacja standardu HTML5*, <http://www.w3.org/TR/CSS/> (stan na dzień 16 lutego 2012)
- [27] *Framework Design: A Role Modeling Approach*, Dirk Riehle, Swiss Federal Institute of Technology, 2000 (stan na dzień 16 lutego 2012)
- [28] *Types and Programming Languages*, Benjamin C. Pierce, 2002
- [29] *Organizacja W3 udostępniająca standardy dla witryn internetowych* <http://www.w3.org/> (stan na dzień 16 lutego 2012)
- [30] *Wypowiedzi znanych programistów, wydawców literatury informatycznej dotyczące Ruby on Rails* <http://www.rubyonrails.pl/cytaty> (stan na dzień 16 lutego 2012)
- [31] *Projekt Basecamp oparty na frameworku Ruby on Rails* <http://basecamphq.com/> (stan na dzień 16 lutego 2012)
- [32] *Język programowania Python* <http://www.python.org> (stan na dzień 16 lutego 2012)

- [33] *Framework Django będący konkurencją dla Ruby on rails, napisany dla języka Python* <http://www.djangoproject.com> (stan na dzień 16 lutego 2012)
- [34] *Narzędzie do tworzenia dokumentacji API RDoc*, <http://rdoc.sourceforge.net> (stan na dzień 16 lutego 2012)
- [35] *Statystyki Google dotyczące zapytań o znane frameworki webowe* <http://www.google.com/insights/search/#cat=5&q=Ruby%20on%20Rails%2CDjango%2CSpring%20MVC&cmpt=q> (stan na dzień 20 grudnia 2010)
- [36] *Schemat pracy w kolejnych iteracjach metodyki Scrum*, [http://effectiveagiledev.com/Portals/0/800px-Scrum\\_process\\_svg.png](http://effectiveagiledev.com/Portals/0/800px-Scrum_process_svg.png) (stan na dzień 4 stycznia 2011)
- [37] *Jedna z najpopularniejszych dystrybucji systemu GNU Linux – Ubuntu* <http://www.ubuntu.com/> (stan na dzień 16 lutego 2012)
- [38] *Narzędzie do trackingu zadań – PivotalTracker*, [www.pivotaltracker.com](http://www.pivotaltracker.com) (stan na dzień 16 lutego 2012)
- [39] *Strona domowa systemu Linux*, <http://www.kernel.org> (stan na dzień 16 lutego 2012)
- [40] *Szablony pozycjonowania CSS – grid layouts*, <http://www.w3.org/TR/css3-grid/> (stan na dzień 16 lutego 2012)
- [41] *Cieniowanie obiektów w CSS – shadows and rounded boxes*, <http://www.w3.org/TR/css3-background/> (stan na dzień 16 lutego 2012)
- [42] *Programowanie zachowawcze*, [http://www.erlang.se/doc/programming\\_rules.shtml#HDR11](http://www.erlang.se/doc/programming_rules.shtml#HDR11) (stan na dzień 16 lutego 2012)