



# COMPONENT-BASED RAILS APPLICATIONS

Large Domains Under Control



STEPHAN HAGEMANN

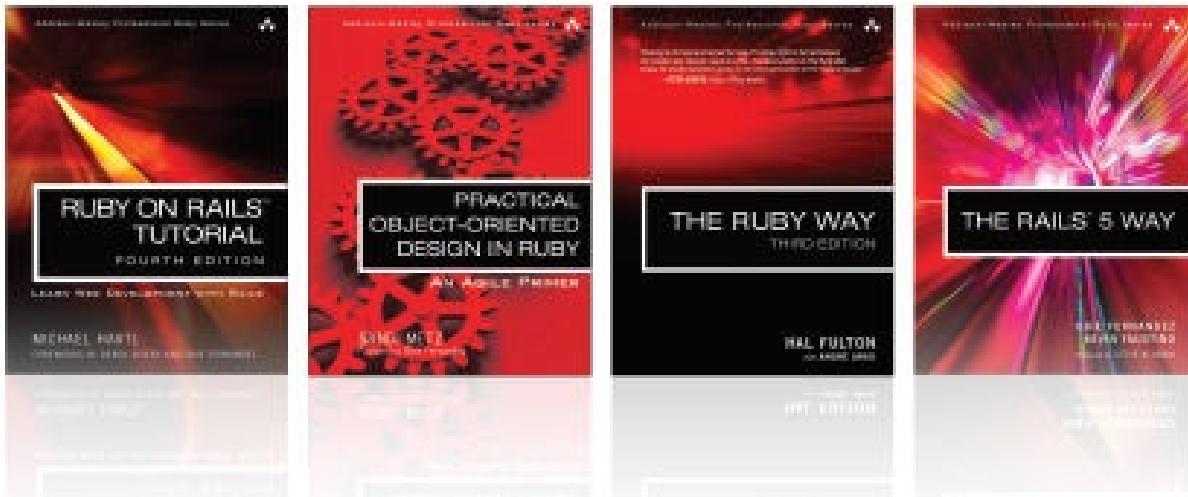
# About This E-Book

---

EPUB is an open, industry-standard format for e-books. However, support for EPUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the e-book in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a "Click here to view code image" link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Pearson Addison-Wesley  
**Professional Ruby Series**  
Obie Fernandez, Series Editor



Visit [informit.com/rubyseries](http://informit.com/rubyseries) for a complete list of available products.

The Pearson Addison-Wesley Professional Ruby Series provides readers with practical, people-oriented, and in-depth information about applying the Ruby platform to create dynamic technology solutions. The series is based on the premise that the need for expert reference books, written by experienced practitioners, will never be satisfied solely by blogs and the Internet.



Make sure to connect with us!  
[informit.com/socialconnect](http://informit.com/socialconnect)

# COMPONENT-BASED RAILS APPLICATIONS

Large Domains Under Control

---

**Stephan Hagemann**

◆ Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town  
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico  
City São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto:intlcs@pearson.com).

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

Library of Congress Control Number: 2018936093

Copyright © 2018 Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearsoned.com/permissions/](http://www.pearsoned.com/permissions/).

ISBN-13: 978-0-13-477458-9

ISBN-10: 0-13-477458-2

# Contents

---

Foreword

Preface

Acknowledgments

About the Author

## **Chapter 1 Introduction to Component-Based Rails Applications**

1.1 Component-Based Rails

    1.1.1 What Component-Based Rails Is

1.2 Benefits of Component-Based Applications

    1.2.1 Improved Communication of Intent

    1.2.2 Improved Collaboration Among Developers

    1.2.3 Improved Creation of Features

    1.2.4 Improved Maintenance of the Application

    1.2.5 Improved Comprehension of Application Parts

    1.2.6 History Repeating?

1.3 Component-Based Ruby

1.4 The Application Continuum

1.5 Related Works

## **Chapter 2 Creating a Component-Based Application**

2.1 The Entire App Inside a Component

2.2 ActiveRecord and Handling Migrations within Components

    2.2.1 Installing Engine Migrations with Rake

    2.2.2 Loading Engine Migrations in Place

### 2.2.3 Known Issues

## 2.3 Handling Dependencies within Components

### 2.3.1 Using path Blocks for Specifying CBRA Dependencies

### 2.3.2 Adding a Regular Gem: slim—Different Templating

### 2.3.3 Locking Down Gem Versions

### 2.3.4 Adding the Development Version of a Gem: Trueskill—A Rating Calculation Library

### 2.3.5 Adding Predictions to the App

## Chapter 3 Tooling for Components

### 3.1 Testing a Component

#### 3.1.1 ActiveRecord Model Specs

#### 3.1.2 Non-ActiveRecord Model Specs

#### 3.1.3 Controller Specs

#### 3.1.4 Helper Specs

#### 3.1.5 Feature Specs

### 3.2 Testing the Main Application

#### 3.2.1 What to Test in the Main App

#### 3.2.2 How to Put Everything Together

#### 3.2.3 How to Set Up a Continuous Integration (CI) Server

### 3.3 Asset Loading for Components

#### 3.3.1 Production-Ready Assets

### 3.4 Switching Databases

### 3.5 Deploying to Platforms-as-a-Service

#### 3.5.1 Deploying to Heroku

#### 3.5.2 Deploying to Pivotal Web Services

### 3.6 Updating Application Dependencies

#### 3.6.1 Updating the Bundle in All Components

#### 3.6.2 Updating Every Component's Gem Version

#### 3.6.3 Updating Rails

#### 3.6.4 Long-Running Dependency Updates

#### 3.6.5 Listing a Dependency Only Once

### 3.7 Proposal for a Different Application Root—Showcasing the Difference of Components' Structure

3.7.1 Why Would I Want to Change That?

3.7.2 How CBRA Can Change How I Present and Perceive My App

3.7.3 Making CI and PaaS Work with the New Structure

3.7.4 What Is Next?

## **Chapter 4 Component Refactorings: Extracting Components out of Components**

4.1 Determining What to Extract: Bottom-Up

4.2 Refactoring: Extract Domain Gem—Predictor

4.2.1 Fixing Inside-Out: Making AppComponent Work Again

4.2.2 Last Step: Ensure That the App Works

4.2.3 Summary of Bottom-Up Component Extractions

4.3 Determining What to Extract: Top-Down

4.3.1 Unclear Dependency Direction

4.3.2 Not Everything Needs Predictor

4.3.3 Implications of the First Set of Extractions

4.3.4 Problem with the Current Solution

4.3.5 Reducing Needed Dependencies

4.4 Refactoring: Pulling Up a UI Component—TeamsAdmin, GamesAdmin, PredictionUI, WelcomeUI

4.4.1 Generate

4.4.2 Move

4.4.3 Fix, Part 1: Reusing Test Helpers

4.4.4 Fix, Part 2: Routing Issues

4.4.5 Fix, Part 3: The Container Application

4.4.6 An Explicitly Contracted Global Navigation

4.5 Refactoring: Pushing Down a Model Component—Teams, Games

4.5.1 Getting Away with Less

4.5.2 Fixing Teams Tests

4.5.3 Fixing Everything Else

4.6 Refactoring: Improve Naming of Component—AppComponent to WebUI

4.6.1 Whatever Happened to AppComponent?

4.6.2 Refactoring Component Names

4.6.3 Refactoring a Component Name Throughout the Application

#### 4.6.4 An Even More Mechanical Approach

### 4.7 More Component Refactoring Patterns

#### 4.7.1 Splitting One Component in Two: Disentangling Unrelated Concepts

#### 4.7.2 API Component

#### 4.7.3 Third-Party Service Adapter

#### 4.7.4 Common Functionality Component

## **Chapter 5 From Ball of Mud to First Components**

### 5.1 Small Steps

### 5.2 One Big Step

#### 5.2.1 Prerequisites

#### 5.2.2 Script This!

#### 5.2.3 A Scripted Refactoring

#### 5.2.4 Cleaning Up Persistence

#### 5.2.5 Using Persistence

#### 5.2.6 Summary

## **Chapter 6 Component-Based Rails in Relation to Other Patterns**

### 6.1 Hexagonal Architecture

#### 6.1.1 Hexagonal Architecture and CBRA

#### 6.1.2 Implementing Hexagonal Rails with CBRA

#### 6.1.3 Teasing Out an Adapter in the Frontend

#### 6.1.4 A Repository for Data Storage

#### 6.1.5 Swappable Data Storage

#### 6.1.6 Summary

### 6.2 Data-Context-Integration (DCI)

#### 6.2.1 DCI and CBRA

#### 6.2.2 Implementing DCI with CBRA

#### 6.2.3 Summary

## **Chapter 7 Components in Other Languages**

### 7.1 Kotlin, Java, and Gradle

### 7.2 .NET / C

### 7.3 Conclusion

## **Appendix**

## A.1 Plain versus --full versus --mountable Engines

A.1.1 Gem

A.1.2 Plain Plugin

A.1.3 Full Engine

A.1.4 Mountable Engine

## A.2 How Do Engine Routes and Engine Mounting Work?

A.2.1 Engines as Subdomains

A.2.2 Engines Are (Unfortunately) Singletons

## A.3 Additional Testing with Older Versions of Rails

A.3.1 View Specs

A.3.2 Routing Specs

## **Index**

# Foreword

---

The other day, I was standing in downtown Mexico admiring one of the city's greatest landmarks, the Torre Latinoamericana, a 44-story skyscraper built in 1956. It's considered an engineering marvel since it's built on very active seismic land. The building's designer, Dr. Leonardo Zeevaert, experienced his creation's performance firsthand during the massive 8.1-scale earthquake that struck the city on September 19, 1985. From his vantage point in his office on the twenty-fifth floor, he witnessed hundreds of buildings collapsing all around the tower.

The building is notable as what might be called a *majestic monolith*, an extraordinary example of a large, complex, and durable self-contained system. David Heinemeier Hansson (DHH) has taken to using the term to describe his favorite architectural pattern for Rails applications. Indeed, he recently bragged on Twitter that Basecamp (the application he designed and extracted Rails from) now exceeds 1,300 controllers, which probably makes it one of the biggest such applications in the world. Nobody outside DHH's team can truly know the quality and durability of that system, but based on the success of their company, we can safely guess that it is pretty good and has survived many an earthquake, so to speak. However, how many of you, no matter how experienced you are with Rails, would like to build one of those systems from scratch? To date, Basecamp has been continually evolved over the course of more than fifteen years. Circa 2005, DHH would have laughed you out of the room if you had told him how big his baby would grow.

Our software industry could be called highly seismic, as it's constantly shaken up by innovation, disruptions, and even outright attacks by malicious actors. Absent the space constraints of constructing physical buildings on expensive real estate, nobody in their right mind wants to start out a software project intending to build the Ruby on Rails equivalent of the Torre Latinoamericana.

Indeed, one of the challenges that faces architects of large Rails project undertakings, whether they actively consider it or not, is whether to steer growth in the mold of David's majestic monolith pattern, or to proactively decompose their system into smaller cooperating systems, often referred to as the *microservices* approach. Undoubtedly, the

latter is trendier, but at the cost of additional complexity that, dare I say, is not merited on most web application projects. If a software team is already productive on Rails, they should probably stick with the monolith approach by default, while staying attentive to opportunities for refactoring functionality out into component services. That's where this book comes in, because at the point where you feel like you can break out a chunk of functionality from your primary codebase, the next step is figuring out the right way to do that.

In *Component-Based Rails Applications*, Stephan builds on a solid foundation of principles laid out by Eric Evans in his seminal work, *Domain-Driven Design*, to describe the use of Ruby gems and Rails engines as a mechanism for implementing components and ensuring explicit dependency management. I found it tremendously interesting in the way that it takes aspects of software design typically associated with large, complex (and distasteful) technologies such as J2EE and integrates them with Rails in a way that stays faithful to Ruby's flexible and friendly nature.

Many of you reading this book have over ten years of experience building Ruby applications. If you've gotten this far, like I have, you're way beyond the basics. You're looking outward, considering how to grow your expertise. As such, this book is a welcome and timely addition to the Addison-Wesley Professional Ruby Series. Time will tell if *CBRA* becomes a modern classic, given the timeless subject matter, but I do feel very confident in how it gives us Ruby pros a comprehensive guide for increasing the sophistication of their designs, without having to forsake the principles of elegance that keep them in our corner of the software world after so many years.

Obie Fernandez  
Mexico City, February 2018

# Preface

---

I am excited for this book to spark a dialogue about application design. My hope is that it will spawn conversations about what is good and what is bad. Conversations about what works and what does not. Conversations about domains and their structure.

## How I Came to Write This Book

It was very shortly after I started learning Ruby and Rails that I had my first professional experience with it. In 2010, I joined the development team at XING, a business social network popular in German-speaking countries. The application we worked on was a large Rails system. It had grown for a couple of years. There were internal services based on HTTP APIs. The deployment process had been refined for years. The various teams were practicing several variants of Agile.

In this situation, I was very happy to have gotten the job and was excited to learn all the Ruby I could and as much Rails as possible. However, I came to a surprising realization: I was not doing much Rails at all. Sure, there were changes to controllers; occasionally there were even new controllers to be written. Sure, we worked with ActiveRecord models, but that was ancillary to what we were trying to achieve for the most part. Really, I was learning about the particular classes and objects that existed in this application. I learned about the way the user API was wrapped by a proxy object. I learned about how my team's part of the application was set up and, to some extent, what the others were doing. In short, I learned even more about the domain of the application than about any technology in particular.

All in all, more than thirty people worked on the two codebases. This was possible without major complications due to how the app was structured. The app was not only split down the middle along the two programming languages, but the Ruby and Rails side also was comprised of around six parts. In these parts, all views, controllers, models, and everything related lived in their own folder structure. The development teams each were responsible for one or more of those parts. User and a few other ActiveRecord models used across the app were in a common place.

Without knowing it at first, I was looking at the closest thing possible to a component-based architecture using Rails 2. One couldn't do much better in Rails 2 because the framework lacked the ability to separate out parts of the application into standalone units that were individually tested with clear responsibilities and trackable expectations about their surroundings.

This all changed later that year with the release of Rails 3. Version 3 meant significant improvements to Rails' internals and the introduction of Rails Engines. This is what engines offer according to the Rails API documentation (<http://api.rubyonrails.org/classes/Rails/Engine.html>): “Rails::Engine allows you to wrap a specific Rails application or subset of functionality and share it with other applications *or within a larger packaged application*. Since Rails 3.0, every Rails::Application is just an engine, which allows for simple feature and application sharing.” [Emphasis added.]

In 2011, I joined Pivotal Labs as a software engineer. The first project I worked on was at a point where the client was about to address a whole new user base. Development was starting on a new part of the application that would ultimately dwarf what existed until then. We discussed a few different approaches and settled on implementing this via engines—three, to be precise: one engine for the existing application, one for the new part, and one for the commonality between the two. Because there were so many unknowns and pitfalls along the way, it took the team a few tries to move the existing application into an engine. In the end, the approach proved successful and more engines were added in the same way.

In 2012, I got to join another project that took the use of engines to the next level. We did not try to have one engine contain the code for one application. Instead, we intended to make sensible components as small as possible. We were basically applying the Single Responsibility Principle (<https://8thlight.com/blog/uncle-bob/2014/05/08/SingleResponsibilityPrinciple.html>) on the level of components. The app ended up having more than thirty components. This led to a whole new way of looking at an application—a much more structural way. Components became the most important mechanism by which to talk about the application. Constant refactorings led the team to have many productive discussions about how to partition and name the pieces. The project was a success and stayed at a very healthy pace for as long as I was able to observe it.

I started publicly talking about this component-based application architecture, first with a lightning talk at the Mountain West Ruby Conference 2012, then with a full talk at Rocky Mountain Ruby 2012 (during which I added the italicized text in the above quote to the documentation of the Rails::Engine class). At some point after that, I found the name that I still use for this kind of application: component-based Rails applications, or CBRA for short, with #cbra as a hashtag and a zebra as the logo. I continued giving talks about this at Mountain West Ruby Conference 2013 as well as the 2014 and 2015 editions of RailsConf.

Rails 4, which was released in 2013, and Rails 5 did not change much in terms of these architectures, and it looks like it will stay that way, so it felt like it was time to write it all up. I hope to be able to explain all the aspects to this architecture in a more consistent way than I can do in blog posts, emails, and tweets.

After announcing my intent at a Boulder Ruby group meetup in early 2014, I started—full of energy—to get all these ideas out of my head in a consistent form. And then I got stuck. Initially, I wanted to write a good introduction, create motivation, continue with general concepts, to then arrive at the inevitable: component-based Rails applications are the way to go, and I would show you how it’s done. It turns out writing it in this deductive style is difficult (for me). The general statements at the beginning better be watertight, in line with research, and probably with a lot of references. That’s work—and I have enough of that during the day! So, sure enough, I stalled over the summer and rethought my approach.

The book you are about to read flips the initial concept on its head—it is much more inductive. I believe that this is not just much easier to write, as I start with what I know best, but I also hope it is a better book, as I focus more on its contribution: to introduce and explain how component-based applications can be written with Rails.

## Who This Book Is For

This book is for people working in Ruby and Rails, comfortable with both the language and the basics of the framework, people who are interested in expanding their horizon for how applications can be structured, Rails applications in particular.

If you are working in an existing, large codebase and are finding that it is hard to implement new features, that the tests are slow, or that you simply have a tough time “finding your way around,” then this book is for you.

If you are working with engineers as part of a team that manages a Ruby/Rails application, then a few parts of this book are for you. While most of it will be too technical if you are not working with the code, some sections go beyond the code to discuss the rationale behind the topics of this book—the discussions of structure are relevant to anyone on a product team as design and features affect application structure. If you are in this situation, just read through [Chapter 1](#).

[Chapters 1](#) and [4](#) are also relevant for engineers working in different languages and frameworks. Chances are that there is some way of representing components in these languages (sometimes it’s more difficult to do, most of the time it is easier) and chances are it would improve the structure of your code there as well.

## What This Book Does Not Cover

While this book covers details of how to create gems and engines, it does not cover the basics of Ruby nor that of the Rails framework.

### Ruby and Rails Basics

This book will not cover the basics of Ruby or Rails. There is a lot of good literature out there, which you should review to get up to speed if the language and the framework themselves are still an issue. Some resources to get you started on both Ruby and Rails include:

- <http://railsbridge.org/> – Rails workshops for women and everyone else

- <http://rubyusergroups.org/> – find your local Ruby group
- <http://rubyconferences.org/> – find your local Ruby conference
- Online resources
  - <https://www.ruby-lang.org/en/documentation/> – extensive list of Ruby resources
  - <https://tutorials.railsapps.org/rails-tutorial> – filterable list of Rails tutorials
  - <http://tryruby.org> – interactive online Ruby REPL
  - <http://guides.rubyonrails.org/> – online handbooks about many Rails topics
  - <http://railsforzombies.org/> – interactive online Rails tutorial and quiz
  - <https://www.railstutorial.org/> – book and screencasts (paid, but free online version of book available)
  - <http://railscasts.com> – Ryan Bates’ screencasts, no longer being updated
- Ruby books
  - *Programming Ruby* (<https://pragprog.com/book/ruby4/programming-ruby-1-9-2-0>) by Dave Thomas, with Chad Fowler and Andy Hunt
  - *Eloquent Ruby* (<http://www.amazon.com/Eloquent-Ruby-Addison-Wesley-Professional/dp/0321584104>) by Russ Olsen
- Rails books
  - *The Rails 5 Way, Fourth Edition* (<https://www.amazon.com/Rails-Way-Addison-Wesley-Professional-Ruby/dp/0134657675>) by Obie Fernandez and Kevin Faustino
  - *Agile Web Development with Rails 4* (<http://www.amazon.com/Agile-Development-Rails-Facets-Ruby/dp/1937785564>) by Sam Ruby, Dave Thomas, and David Heinemeier Hansson

## Rails Version 2

This book will not cover Rails 2 applications specifically. This is because Rails Engines, the main implementation vehicle for components in Rails, only came into existence with Rails 3. If you are on Rails 2, the chapters that are still interesting for you are exactly the same as those for non-Rails Ruby applications: Chapters 2 and 5 are more applicable to Rails 3, 4, and 5 applications. Chapters 4, 6, and 6 contain example code that is Rails 5, but contain many parts relevant to Rails 2 as well.

## How This Book Is Structured

The next chapter, Chapter 1, Introduction to Component-Based Rails Applications, presents in broad strokes the topic of this book. It puts it into context of Rails, previous work, and research.

Chapter 2, Creating a Component-Based Application, tells the story of creating a full Rails

application within a component. From the first steps to migrations and dependency management, it covers the common pitfalls of the unavoidable aspects of component-based Rails.

[Chapter 3](#), Tooling for Components, explains how to test component-based applications, how to manage assets, long-term dependency management, and how to deploy to PaaS.

To start, [Chapter 4](#), Component Refactorings: Extracting Components out of Components, takes a step back to analyze why component-based architectures provide benefits to the overall application structure. Then, it discusses how to identify component seams and the fundamental ways of extracting components within existing systems.

[Chapter 5](#), From Ball of Mud to First Components, takes the lessons from the previous chapters and applies them to what is arguably the most difficult step in the introduction of CBRA: the start. A scripted approach that iterates over the necessary refactoring works for applications of any size.

With [Chapter 6](#), Component-Based Rails in Relation to Other Patterns, we attempt to show how CBRA fits with two popular structural patterns: Hexagonal architecture and Data-Context-Interaction architecture. It turns out that the contribution of CBRA in conjunction with these approaches is the enforcement of structure, dependencies, and boundaries.

[Chapter 7](#), Components in Other Languages, closes out the book by pointing the reader toward approaches to components in other frameworks and programming languages.

Register your copy of *Component-Based Rails Applications* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to [informit.com/register](http://informit.com/register) and log in or create an account. Enter the product ISBN (9780134774589) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

# Acknowledgments

---

No matter *how* I wrote this book, it certainly would not have gotten written at all without the help of many.

First, I would like to thank the reviewers who helped me improve on previous versions of the book: Barrett Clark, Paul Ort, Tammer Saleh, and Obie Fernandez. I would like to thank Chris Zahn for making sure that the book meets Pearson's publishing standards. Big thanks go to Debra Williams Cauley for working with me on the entire publishing process over the past year. Thanks to Julie Nahil and Dhayanidhi Karunanidhi for getting the book produced and proofed.

I would like to thank all those discussing component-based Rails applications with me on conferences and especially in the CBRA discussion group (<https://groups.google.com/forum/#topic/components-in-rails>). Many questions there have led me to continue to explore the notion of CBRA and its implications. To the many folks I worked with at the Pivotal Labs' Boulder office—especially Lew Parker, Georg Apitz, Austin Vance, Arjun Sharma, David Farber, Damien LeBerrigaud, Mike Gehard, Jeff Dean, Charles LeRose, Ben Smith, and Enrico Teotti—thank you for many fruitful discussions. Mike Barinek, my former manager in the Boulder office, was always available as a discussion partner and guide. Thanks for many exciting coding and whiteboarding sessions! My fellow contributors of previous versions of this book, Ryan Platte, Enrico Teotti, and Ben Smith, spearheaded their own directions in component-based Rails. The exchanges with them have made me able to create a better result.

Finally, I would like to thank Sarah Tipton, my wife, for listening, being patient, and for reading through many drafts.

## About the Author

---

**Stephan Hagemann** most recently has been supporting clients of Pivotal Software in journeys of digital transformation. In these, organizational change is as much a topic as the implementation of Agile processes and software system architecture. This work grew out of Stephan's previous management and engineering roles within Pivotal Labs, Pivotal's software consulting group. Prior to joining Pivotal, Stephan moved to the United States from Germany where he worked as a software engineer for XING. He has a doctorate in information systems from the University of Münster in Germany.

“A good architecture maximizes the number of decisions not made. It’s the purpose of an architecture to prevent you from making decisions. It is to keep your options open for as long as those options can be open.”

—From Robert Martin’s Ruby Midwest 2011 keynote, “Architecture the Lost Years.”  
<https://www.youtube.com/watch?v=WpkDN78P884> at 57:18

“Intelligence tries to maximize future freedom of action and keep options open.”

—From Alex Wissner-Gross’s TED talk, “A New Equation for Intelligence.”  
[http://www.ted.com/talks/alex\\_wissner\\_gross\\_a\\_new\\_equation\\_for\\_intelligence](http://www.ted.com/talks/alex_wissner_gross_a_new_equation_for_intelligence) at 5:18

“I like to draw these lines. Architecture is the art of drawing lines. With the interesting rule that once you have drawn the lines, all the dependencies that cross that line go in the same direction.”

—From Robert Martin’s Ruby Midwest 2011 keynote, “Architecture the Lost Years.”  
<https://www.youtube.com/watch?v=WpkDN78P884> at 43:20

# CHAPTER 1

---

## Introduction to Component-Based Rails Applications



Photo: Syndromeda/Shutterstock

In software engineering, components encompass cohesive pieces of functions and data to encapsulate reusable business functionality ([http://heim.ifi.uio.no/~frank/inf5040/CBSE/Component-Based\\_Software\\_Engineering\\_-\\_ch1.pdf](http://heim.ifi.uio.no/~frank/inf5040/CBSE/Component-Based_Software_Engineering_-_ch1.pdf)). To this end, components can use other components by including them as dependencies and accessing their functionality and data.

Interestingly, the first sentence of this definition is also the typical definition given for objects in object-oriented software design (<https://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep#Object>). The two concepts are related, indeed. In practice, a component is typically made up of one or more object definitions.

The difference between objects (and the classes that define them) and components is that objects can commonly interact with each other without specifying a dependency explicitly. On the contrary, components can only depend on each other and interact with each other if

they state their dependency explicitly in some way. That in turn means that if one component depends on another, the second component cannot depend on the first. Therefore, objects can be part of circular dependencies while components cannot.

In short, the biggest difference between components and objects is that the former are much more about specifying dependencies than the latter.

## 1.1 Component-Based Rails

The thought of components in Rails is odd to many: Is Rails not good enough? Do you want to turn it into Java? These questions are asked when you propose that there might be more to architecture than Rails' version of the model-view-controller (MVC) pattern. More precisely, these questions were commonly asked in the late 2000s. The more the community around Rails grew up—and the more applications built with it were being put to the test in different situations—the more common voices became that were looking for solutions to problems they hadn't faced before.

The need for components in Rails arises because, fundamentally, MVC does not solve all problems.

Then, there is the reaction and the questions that come up when one explains how to implement component-based architecture in Rails based on gems and engines. But gems are for reuse, aren't they? And engines are for extracting parts of the app that you don't really care about, no?

### 1.1.1 What Component-Based Rails Is

So, what is component-based Rails? It is applying component-based software design to Rails, with gems and engines as the vehicles of implementing components and ensuring explicit dependency management. It is a common misconception that gems and engines are for packaging, distribution, and reuse only.

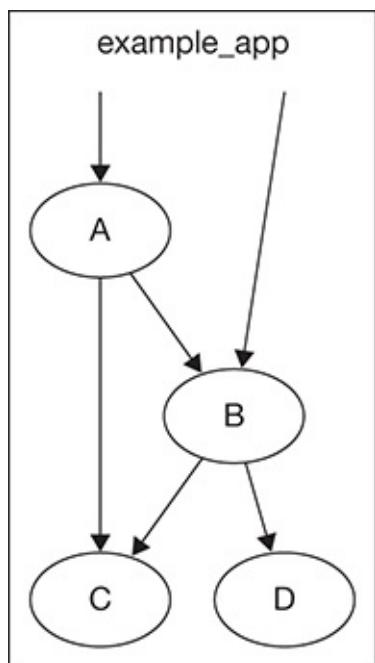
What is an engine? According to RailsGuides – Getting Started with Engines (<http://guides.rubyonrails.org/engines.html>), “[e]ngines can be considered miniature applications that provide functionality to their host applications. A Rails application is actually just a ‘supercharged’ engine, with the `Rails::Application` class inheriting a lot of its behavior from `Rails::Engine`.<sup>1</sup>” RailsGuides suggests some example engines that you can check out: “Devise (<https://github.com/plataformatec/devise>), an engine that provides authentication for its parent applications, or Thredded (<https://github.com/thredded/thredded>), an engine that provides forum functionality. There’s also Spree (<https://github.com/spree/spree>) which provides an e-commerce platform, and RefineryCMS (<https://github.com/refinery/refinerycms>), a CMS engine.”

Via this vehicle, we can create applications comprised of components that:

- Are isolated
- Are individually testable
- Are explicitly dependent on each other

- And create non-cyclic dependency graphs

Throughout this book, we will be using component diagrams to illustrate which components exist in an application and how they depend on each other. [Figure 1.1](#) is a component diagram for an example application that contains the components A, B, C, and D. Every component is depicted as an ellipse, with its name written in it. The different sizes of the ellipses aren't meaningful; the ellipses are sized to accommodate their contents. Arrows denote dependencies between components. An arrow pointing from A to B means that “A is dependent on B.” For Rails applications, this means that there is an entry for the gem B in A's `gemspec`. The name of the application, `example_app` in this case, is given at the top. When it is present, there will also be a bounding box denoting the application that contains the components. Arrows pointing from the application name to components, like the arrows going into A and B in this case, denote direct dependencies of the application.



**Figure 1.1.** Component diagram example

You can generate these component diagrams using the `cobradeps` gem (`gem install cobradeps`), whose source code is available at <https://github.com/shageman/cobradeps>. Just install it and point it at the root directory of your component-based Rails application, like so: `cobradeps -g myapp path/to/app`.

Due to Ruby's flexible language constructs, there are many ways in which we as Rails developers can implement component-based architectures. From simple vertically separated Rails apps modularized by engines to fully service-enabled, deep component-based architectures, many variants are possible. Even if the previous statement makes no sense to you, this book will introduce you to how you can take your first steps with components in Rails. You will also learn to determine the extent of component-based Rails which is best for you.

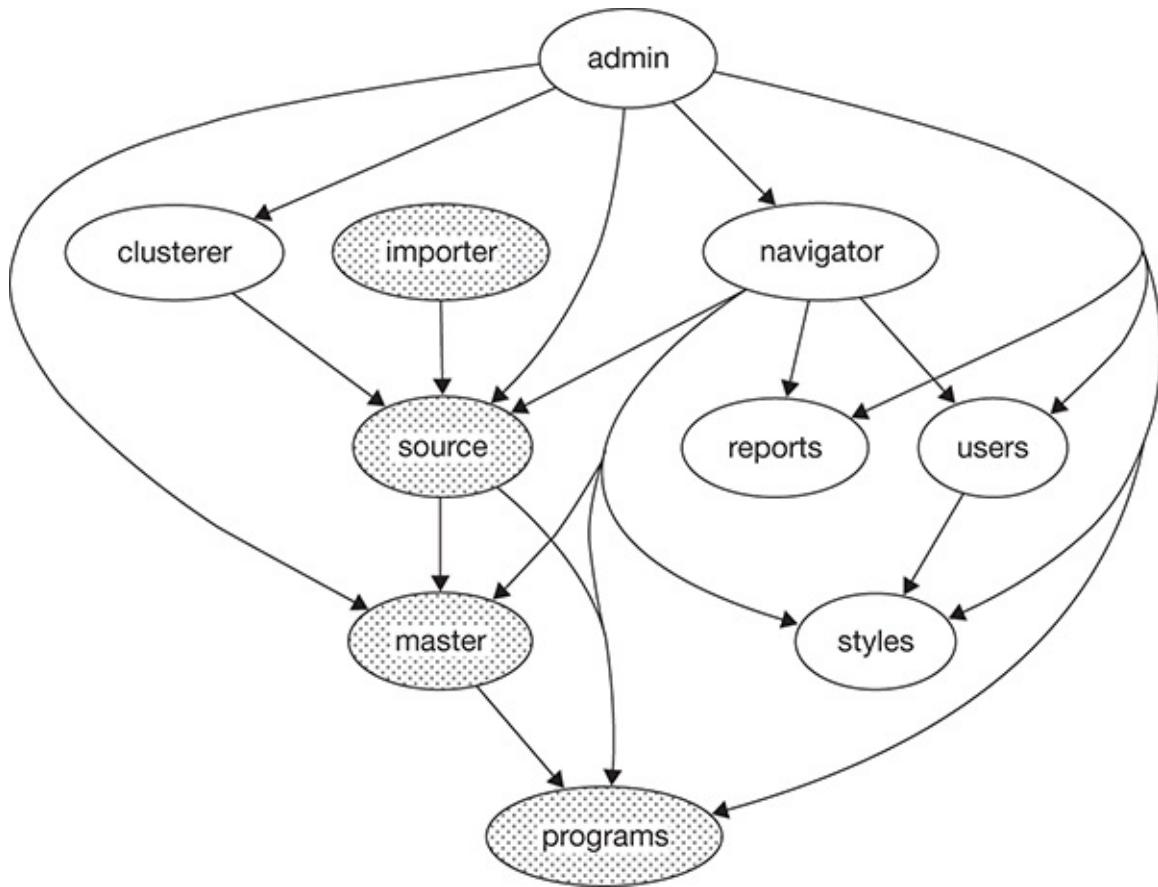
## 1.2 Benefits of Component-Based Applications

This section discusses the details of what we achieve by creating truly component-based applications.

### 1.2.1 Improved Communication of Intent

When multiple components are present, they make large parts of our application visible on a high level. They force these parts to be named, triggered by discussions and continuous refactorings by engineers. Their naming, their contents, and their structure trigger more discussions throughout their lifecycle. On top of that, the dependencies between each other become explicit.

Figure 1.2 shows the dependency graph of a fictional application that is similar to an application I once worked on.



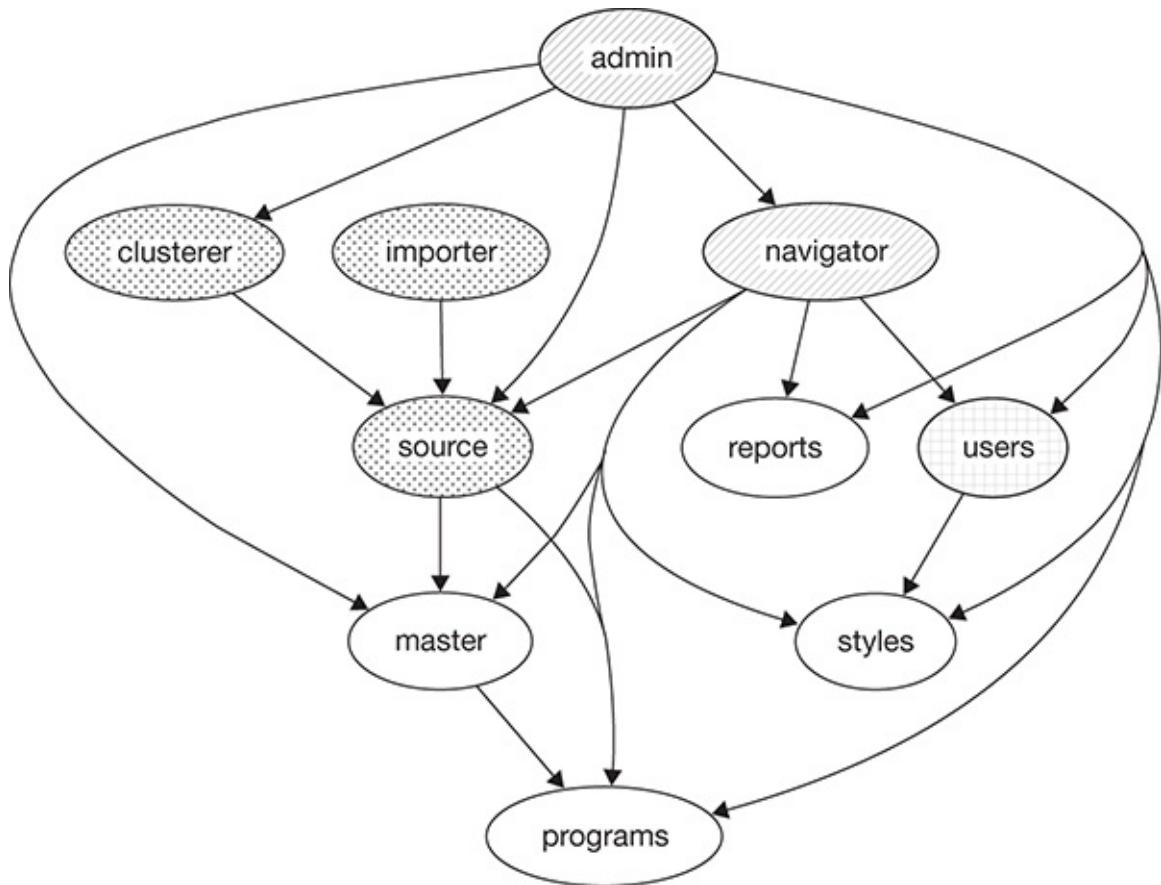
**Figure 1.2.** Sample dependency graph of a component-based application

In Figure 1.2, the highlighted areas show a dependency chain between importer, source, master, and program. Even if we learn no more about this application, we know that there is an *importer* component that is concerned with importing (from?) sources. It might be importing *programs* or it might be importing something into *programs*. It also looks like some sort of cleanup might be happening from *sources* to *master* records. Someone with knowledge of the system could use these high-level concepts and their relationships to explain quickly to us what is really going on and where our initial impressions about the application are right or wrong.

## 1.2.2 Improved Collaboration Among Developers

As the parts of the application become explicit through components, collaboration also improves. First, it is easier to communicate about which parts of the codebase a person, pair, or team is working on. As dependencies are explicit, others know whether they are affected by this work at all or whether they are completely independent.

In the sample graph depicted in [Figure 1.3](#), for example, if one developer is working on *source*, they need to run tests for all dotted components (clusterer, importer, source) and striped components (admin, navigator). If someone else were to work on *user* at the same time, they would need to run all gridded components (users) and striped components (as before). In this scenario, the only possible points of conflict are the striped components: *navigator* and *admin*. Through components, the area of potential conflict has been reduced to 20 percent in comparison to a monolithic app (if we assume components are of equal size).



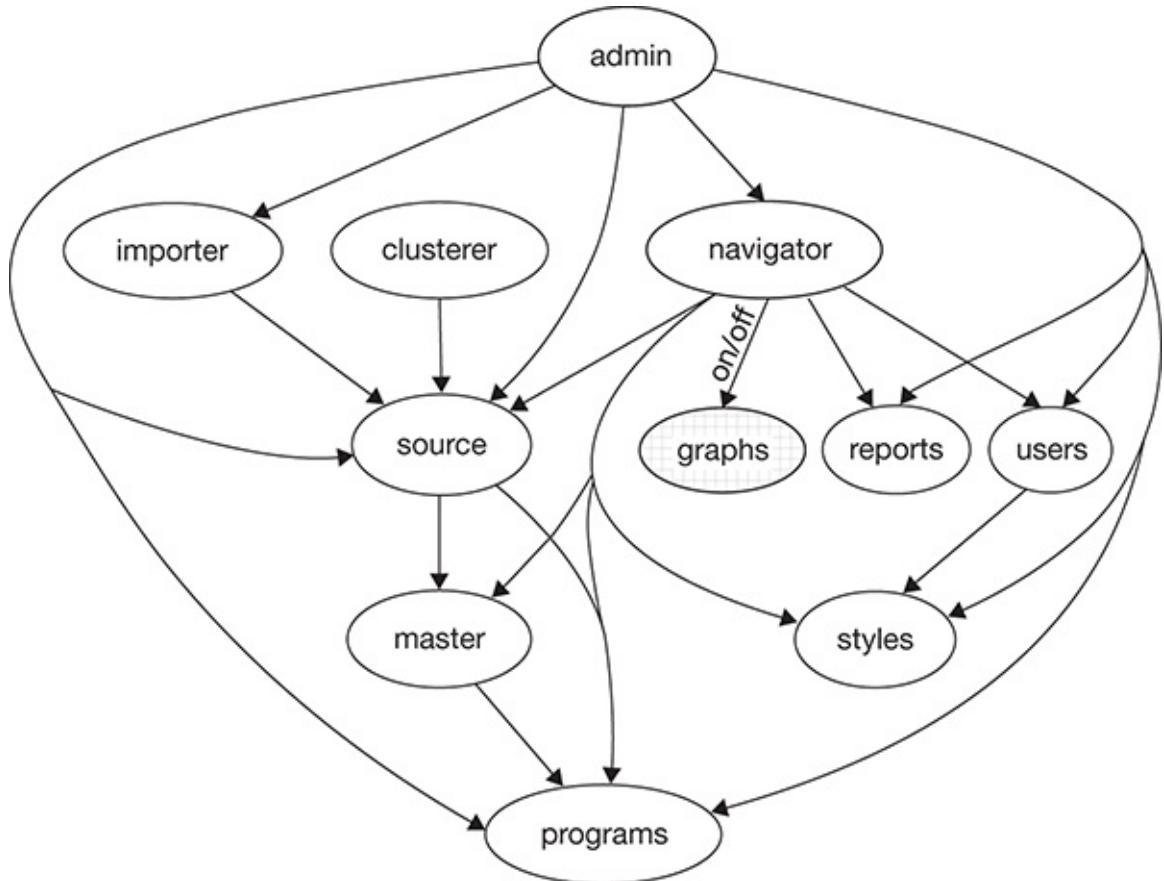
**Figure 1.3.** Possible conflict when working on separate parts of the system (potential conflicts are in the admin and navigator components)

As components evolve, work can—but does not have to be—more specialized. There might be components that are purely front-end, mainly front-end, domain-only, persistence-focused, or concerned only with third-party APIs. Knowledge existing around these parts of the system can easily be documented in its canonical place: the README of the component. This might include difficulties, design decisions, coding conventions specific to a component, or a list of people to talk to about the respective component.

### 1.2.3 Improved Creation of Features

Components also help improve the way the creation of new features within an application is happening.

If a new feature is developed as a new component (like *graphs* in [Figure 1.4](#)), the entire component can be feature-flagged *not* to go live in production until it is approved by the product manager. No source control branching, feature-flagging gems, or anything else is necessary—simply switch the gem’s loading off in the production environment.



**Figure 1.4.** New components can be easily turned on and off

The presence of components forces thinking in terms of distinct parts. What belongs together and what does not? This triggers conversations about higher-level structure and architecture. Once working in components becomes more commonplace, the overhead of creating a new component—splitting one into two, or merging two together—is greatly reduced. What the team ends up with is an application that can be developed faster and better, which brings us to maintenance.

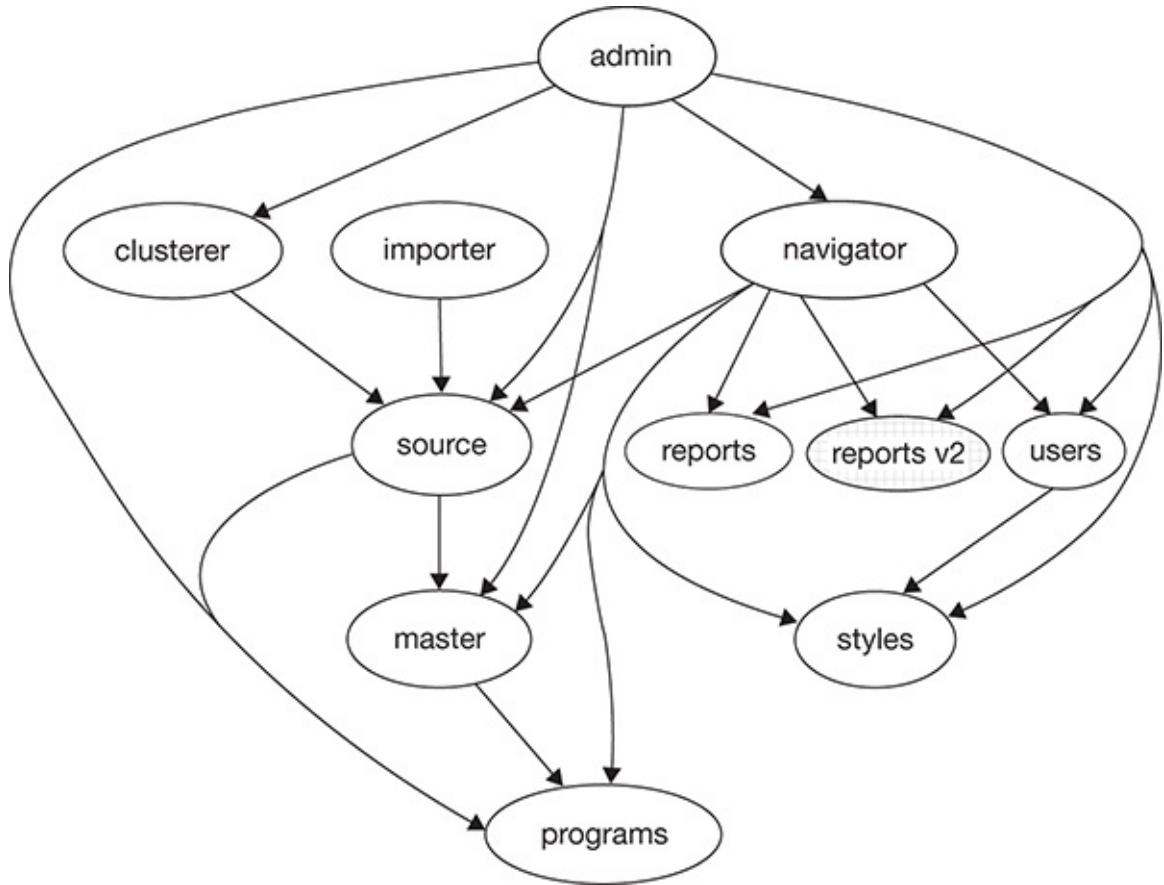
### 1.2.4 Improved Maintenance of the Application

Long-term maintenance is greatly improved with components. Imagine that *reports* contains an API and that a new version has to be written (see [Figure 1.5](#)). One way of solving this with components is to:

- Duplicate the existing *reports* engine
- Rename it to *reports v2* in the process
- Mount it at a different place

- And start making the required changes (without ever affecting the functionality of the existing API in *reports*)

When *reports* can be sunsetted, it is simply removed from the routes, it is no longer loaded, and the gem code is deleted.

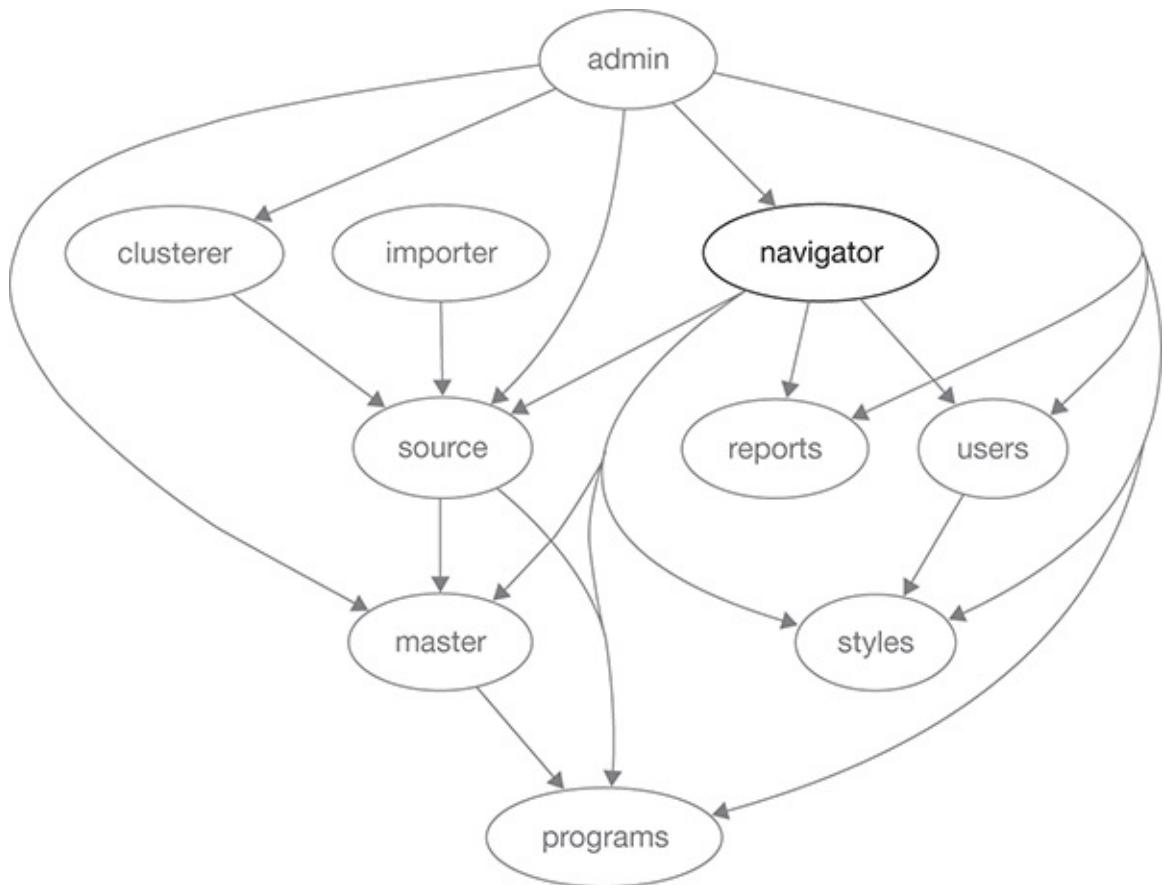


**Figure 1.5.** Improved maintenance

Since the exact dependencies of every component are recorded in their respective gem specifications, upgrading of external dependencies is easier. Updating the entire application to a new version of Rails no longer needs to be a monolithic commit. Individual components can be tested and adapted to work with the new version of Rails (while maintaining compatibility with the current version), all the while continually committing to the main development branch. Once *all* components are compatible with the new version of Rails, the entire app switches over, at which point the tests for the older version of Rails can be removed.

### 1.2.5 Improved Comprehension of Application Parts

Finally, there is an underlying benefit to the comprehensibility of the codebase when it is split into components. This stems from splitting a larger structure into smaller constituents that are connected through much smaller interfaces, which allows developers to ignore other parts of the application while they are working on one component, knowing that they are looking at a highly cohesive subset of a larger entity (see [Figure 1.6](#)).



**Figure 1.6.** Improved comprehension

To explain this further, let's take a little detour.

For social networks, there are two often cited relationships between the number of people in a network and the utility that the network provides. The first one is Metcalfe's law (<https://blog.simeonov.com/2006/07/26/metcalfe-s-law-more-misunderstood-than-wrong/>), which is a statement about the potential benefit when equated to the number of point-to-point connections possible in the network. This number grows proportionally to  $n^2$ , with  $n$  being the number of people, or nodes. That is, for every node added, there are as many new connections possible as there were already nodes in the network.

The second one is Reed's law (<https://www.networkworld.com/article/2225509/cisco-subnet/understand-and-obey-the-laws-of-networking.html>), which makes a similar statement but uses the number of subgroups that could possibly be formed. This relationship grows much more rapidly, being proportional to  $2^n$ . That is, for every node added, the number of subgroups doubles. This is easy to see when analyzing how possible subgroups change when a node is added. When node  $X$  is added, all the previously possible subgroups are still possible. However, in addition to those groups, all the same subgroups—just with  $X$  added—are possible. Both Metcalfe's and Reed's laws can be summarized as positive network effects: The utility of a network grows superlinearly with the growth of its nodes.

What are the nodes in a codebase? Let us say that the nodes are classes. Then, the connections and subgroups are classes as they are interacting. If we accept this idea, what is the utility of this network? One could describe it as the possibility of the classes being combined to provide complex functionality. I am not sure which one of the two network laws fits our use case better. I am leaning toward Reed's law, as not all interactions

between classes involve only two classes. However, they will typically also not involve many more than a few classes. In effect, the “likely utility” will lie somewhere between  $n^2$  and  $2^n$ .

So, the more classes the better? No. There is one big, *big* difference between the formation of social networks and code networks: We would like to be in control of the networks of classes we create. In fact, for the long-term maintainability of our codebases, we *need* to stay in control. And our ability to control our application is directly connected to its ability to outsmart us, that is, for something unexpected to happen. The less complex a system is, the more control we retain and the less the unexpected can happen. So, for our purposes, *utility* (as discussed previously) equals *complexity*.

The categories of improvements we have discussed all result from the abilities of components to reduce the complexity within an application by creating structure. For a system to stay manageable, there should be fewer interactions possible, not more.

By adding components, we change the network of classes and are actively countering the effects of network effects. As an example, let us assume an application has some number,  $n$ , classes, which can all interact because there are no boundaries. In comparison, let us assume that the same system can also be built as two components that have  $X/2$  nodes each, but that need an additional node for the communication between the components. [Tables 1.1](#) and [1.2](#) list the complexities as calculated by Metcalfe’s and Reed’s observations, respectively.<sup>1</sup>

**Table 1.1** Network complexities according to Metcalfe’s law

<b><i>n</i></b>	<b>1 App (Metcalfe)</b>	<b>2 Components (Metcalfe)</b>	<b>Comparison</b>
2	4	3	75.00%
10	100	51	51.00%
50	2,500	1,251	50.04%
100	10,000	5,001	50.01%

**Table 1.2** Network complexities according to Reed’s law

<b><i>n</i></b>	<b>1 App (Reed)</b>	<b>2 Components (Reed)</b>	<b>Comparison</b>
2		4	125.00%
10		1,024	6.35%
50	1,125,899,906,842,620	67,108,865	0.00%
100	1,267,650,600,228,230,000,000,000,000,000		0.00%

Metcalfe's law judges the two-component solution to be about half as complex as the monolithic application solution. Reed's law, however, judges the two solutions to be so vastly different that a comparison makes almost no sense: A large monolithic application is much more complex than one with two components.

As said before, the truth probably lies somewhere in the middle. However, there are two more things to take note of in these tables:

1. The top value for Reed's law shows the two-component solution to be more complex than the monolithic app at 125 percent. Introducing components creates overhead! There are situations in which this overhead is not outweighed by the benefits of the improved structure. For the preceding formulas, simply adding one more node to create a three-node example will make the component architecture outperform the monolith. In a real application, this inverted relationship may hold for larger applications, for several reasons.
  - The split is not as optimal (similarly sized components show the largest improvements in these examples)
  - The interfaces are not quite as clean and more classes are necessary to connect the two components
2. The columns for the two-component solution judged with either metric still show these applications' complexity to grow extremely quickly. Overall, adding more code to an application will always increase its potential complexity, whether or not quality architecture is applied. It is the nature of complex systems that they are just that: *complex*. We apply high-quality architecture, and break down problems into smaller pieces simply to stay afloat!

This last observation may seem depressing. After all, why try to fix what is not fixable? There is even research that uncovers the size of codebases as the only relevant factor in the amount of bugs found in a system (*The Confounding Effect of Class Size on the Validity of Object-oriented Metrics* [1999], <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.2141>, by Council Canada et al.). However, I believe that the verdict is still out on whether there are any factors that push down the average number of bugs for any given codebase size. And I, for one, believe that a better organized system will reduce the factor by which we introduce bugs.

### 1.2.6 History Repeating?

One point raised sometimes as a question, sometimes as criticism, is: Isn't this all just Java all over again? More than statements about Java, comments like these are referring to perceived problems with Java applications, and especially Java Platform Enterprise Edition (Java EE) applications: Is this not too many classes, too many interfaces, too many packages, too much code, too many patterns?

A *pattern* is the pair of problem *and* solution. If you have no problem with your app, component-based Rails applications (CBRA) is not your solution. However, problems

come in a wide variety:

- Feature development has become more expensive.
- Test suites have become slow, or brittle, or have been abandoned.
- Bugs are becoming increasingly difficult to track down.

If your problems are related to the sheer size of the application, then structural software patterns probably have an answer, and CBRA might be a solution to your problems.

## 1.3 Component-Based Ruby

While this book is called *Component-Based Rails*, the techniques used apply to any other Ruby application.

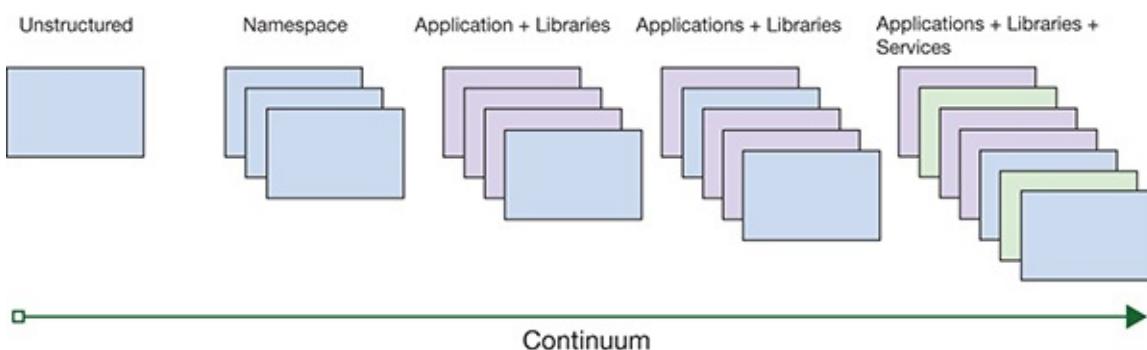
There are a few reasons for picking *Rails* for the title of this book. To start, Rails as a framework has been powerful in shaping the Ruby ecosystem. Also, in practice, the largest Ruby codebases (and thus the ones benefiting the most from thoughtful structure) seem to also be Rails applications.

If you are reading this book as a non-Rails Ruby developer or with a Ruby-not-Rails application in mind, please keep reading. Depending on whether you are using none or some of Rails' libraries, you might find some chapters more interesting than others. Chapters 2, 3, and 4, while relevant, are largely focused on Rails applications. Chapters 5 and 6 contain examples with Rails code, but are relevant beyond.

## 1.4 The Application Continuum

Adequately managing the lifecycle of software applications through growth and change is an ongoing challenge for every practitioner. It is a truism that one size does not fit all when it comes to software and architecture. More specifically, it can be said that the maturity of a domain—or rather, our understanding and software representation of a domain—greatly impacts the structure of an application.

Mike Barinek formalized this notion with the Application Continuum and made it even more specific (<http://www.appcontinuum.io/>). His key observation is that different levels of domain knowledge should make us favor less or more structured approaches to our software (see Figure 1.7).



**Figure 1.7.** The application continuum

On the far left of the continuum is code that is completely unstructured on a higher level.

This would only be used if we knew nothing about the domain and it is likely too early for us to write any software. At this stage, one should consider investing into the understanding of the domain using techniques that are geared directly at information gathering and knowledge generation. *Lean Startup* (<http://theleanstartup.com/>) by Eric Ries and *Lean UX* (<http://www.jeffgothelf.com/lean-ux-book/>) by Jeff Gothelf and Josh Seiden are recent books covering this kind of work.

On the far right of the continuum is a well-structured software system that employs multiple applications with components using libraries for common functionality, acting as services or end-user applications.

In the middle of the continuum lie component-based application architectures. They are a natural midpoint in that they allow for the creation of structure while still allowing for simple deployments and fast refactorings. These things are achieved because all structure is created within *one* application.

The continuum can also be viewed as the starting point of “application cycles.” As we move an application from left to right through the continuum and it becomes more structured, it can also be split into multiple applications. On the level of one of those applications, we might have to “cycle” back around (or move toward the left on the continuum) to a less structured approach as we face new uncertainties.

## 1.5 Related Works

There are a few other works that have dealt with growing Rails apps and how to tackle related problems, which you should check out.

- Talks
  - Simple Made Easy (<http://www.infoq.com/presentations/Simple-Made-Easy>) at strangloop 2011 by Rich Hickey
  - Architecture the Lost Years ([https://www.youtube.com/watch?v=WpkDN78P884&list=TLfXq\\_rdLkeVs](https://www.youtube.com/watch?v=WpkDN78P884&list=TLfXq_rdLkeVs)) keynote at Ruby Midwest 2011 by Robert Martin
  - Carson: On the Path from Big-Ball-of-Mud to SOA (<http://confreaks.com/videos/1234-aloharuby2012-carson-on-the-path-from-big-ball-of-mud-to-soa>) at Aloha Ruby Conf 2012 by James Rosen
  - Facing the Monolith: Overcoming Monolithic Applications with SOA (<http://confreaks.com/videos/1249-aloharuby2012-facing-the-monolith-overcoming-monolithic-applications-with-soa>) at Aloha Ruby Conf 2012 by Charles Max Wood
  - Hexagonal Rails (<http://confreaks.com/videos/977-goruco2012-hexagonal-rails>) GORUCO 2012 by Matt Wynne
  - How I Architected My Big Rails App for Success! (<http://www.youtube.com/watch?v=uDaBtqEYNBo>) at Rocky Mountain Ruby 2013 by Ben Smith

- SOA Without the Tears (<http://confreaks.com/videos/2675-gogaruco2013-soa-without-the-tears>) at Golden Gate Ruby Conference 2013 by John Wilkinson and Anthony Zacharakis
- Building Complex Domains in Rails (<http://www.confreaks.com/videos/4177-rmr2014-build-complex-domains-in-rails>) at Rocky Mountain Ruby 2014 by Mike AbiEzzi
- Domain Driven Design and Hexagonal Architecture with Rails (<http://confreaks.com/videos/3316-railsconf-domain-driven-design-and-hexagonal-architecture-with-rails>) at RailsConf 2014 by Declan Whelan, Eric Roberts

- Blog Posts

- Rails 4 Engines (<http://tech.taskrabbit.com/blog/2014/02/11/rails-4-engines/>) by Brian Leonard
- 7 Patterns to Refactor Fat ActiveRecord Models (<http://blog.codeclimate.com/blog/2012/10/17/7-ways-to-decompose-fat-activerecord-models/>) by Bryan Helmkamp
- A Conversation with Badri Janakiraman about Hexagonal Rails (<http://martinfowler.com/articles/badri-hexagonal/>) by Martin Fowler

- Books

- Enterprise Rails (<http://www.amazon.com/Enterprise-Rails-Dan-Chak/dp/0596515200>) by Dan Chak
- Growing Rails Applications in Practice (<https://leanpub.com/growing-rails>) by Henning Koch and Thomas Eisenbarth

---

1. Michael J. Gunther, *Auftragstaktik: The Basis for Modern Military Command?* (Fort Leavenworth, KS: School of Advanced Military Studies, U.S. Army Command and General Staff College, 2012), 3 (emphasis added).

# CHAPTER 2

---

## Creating a Component-Based Application



Photo: mattiaath/123RF

In preparation for creating your first CBRA app, you should ensure that your system is up to date with regard to Ruby and Rails. I suggest using the latest published versions of both Ruby and the Rails gems. The way I have my system set up, using the Ruby Version Manager (RVM) (<http://rvm.io/>), is something like the following:

**Install rvm, bundler, and rails. Execute anywhere**

[Click here to view code image](#)

```
$ rvm get stable  
$ rvm install 2.4.2  
$ gem install bundler -v '1.15.4'  
$ gem install rails -v '5.1.4'
```

---

### Code Samples: The Sportsball App

Throughout this book, we are going to use the example of a Sportsball app.

Obviously from the realm of sports, this app will allow us to store and manage teams and games, and analyze performance. The application will prove to be simple. It is simple enough that it does not overwhelm the explanations of the book with specific domain knowledge. It will sometimes feel too simple, as the overhead from componentization will at times far outweigh the benefits we can reap within such a small domain. In those cases, we will discuss that along with the respective topic.

All references to directories are relative to the root of the Sportsball application. In this spirit, `./` also refers to the root of the app.

Commands that are to be typed in are prefixed with a `$` sign. The respective code listing will specify in which directory to execute the command. For this chapter, only the commands and source code changes to be executed will be exhaustive, meaning you only need the book. To do the same for the subsequent chapters would drown out the actual topics with too much noise. For all chapters, please refer to the source code available for this book at <https://github.com/shageman/component-based-rails-applications-book>. The source code is organized chapter by chapter and contains a complete list of commands and changes.

---

## 2.1 The Entire App Inside a Component

In this section, we will be creating a Rails application that contains no handwritten code directly in the `app` folder and instead has all code inside an engine mounted into the application.

The first step in creating a CBRA app is the same as creating any new Rails app: `rails new!`

### Generate Sportsball app. Execute where you like to store your Rails projects

[Click here to view code image](#)

```
$ rails new sportsball  
$ cd sportsball
```

I often refer to this Rails app as the *main app* or *container app*. However, we are not going to bother writing any code of our application into the newly created folder structure. Instead, we are going to write all of it inside a component. That component will be implemented through a Rails engine. In fact, let us force ourselves to stick to this and delete the `app` folder.

### Delete the app folder. Execute in `./`

```
$ rm -rf app
```

---

### Why Would You Delete the app Folder?

It is optional to delete the `app` folder at this point. I am suggesting it for a couple of reasons. For one, it sends a clear message to everyone working on the codebase that something special is going on here. Also, the `app` folder is not currently needed and is likely not going to be needed in the future. Thus, it would be cruft to keep it

around right now. And it can always be added back later if needed.

---

## Sprockets 4 Forces Existence of app Folder

As discussed in the sprockets-rails gem issue 369 (<https://github.com/rails/sprockets-rails/issues/369>), sprockets version 4 forces the existence of a configuration file in `app/assets/config/manifest.js`. This file is sprockets' new mechanism for configuring which assets to include in precompilation (<http://eileencodes.com/posts/the-sprockets-4-manifest/>). The introduction of this change is the first time since Rails 3 that the existence of an `app` folder is forced for Rails applications.

Some release candidate versions of Rails 5.1 (specifically Rails 5.1.0.rc1) installed sprockets 4.0.0.beta4, while Rails 5.1.0.rc2 installed the older version sprockets 3.7.1. At the time of writing, Rails 5.1.4 installs sprockets 3.7.1. Let's hope that the configurability of the precompilation configuration comes back to sprockets before Rails moves to newer versions for good.

---

There is no obvious candidate as to where the components of our application should be added in the created folder structure. All folders have their specific purpose and our components don't really fit in any of them. Consequently, I believe components should not go into the created folder structure and instead should find their place in a new directory under the root. I like to use `components` as the folder name.

### Create the `components` folder. Execute in ./

```
$ mkdir components
```

To create our first component, we will use the `rails plugin new` command to create a Rails engine. For lack of a better name, we will call this first component `app_component`.

---

## Pick Good Component Names!

Component names should be picked carefully.

A first technical guideline is that specific names are better than unspecific names. If erring on one side, err on over-specifying names. This is because refactorings from more specific names to less specific names tend to be easy, while the inverse is not true.

A second technical guideline is to avoid naming collisions in general (homonyms) and with names commonly used in Rails applications specifically. Even if for the underlying concept it would make a specific name, if homonyms are created, it is not a good choice. The following is a list of commonly chosen but particularly bad names for these reasons:

- App
- Component

- Engine
- Common
- General
- Base
- Core
- Misc
- Context

These names are too unspecific, too broad. Also, there are no situations in which one would say, “Oh, this object shouldn’t live in this component—it doesn’t even fit to the concept described by its name!” That’s because these names don’t create a meaningful context for their contents. Thus, the last (and arguably most important) guideline for finding the right name is to express the specific contribution a component makes within an application. The following are a few examples of names that do not follow this guideline.

- BaseComponent
- CoreGem
- <your company/division/group name>
- <the abbreviation of your company/division/group name>

---

Initially, I will use `app_component` as a component name. This is despite what I said before about the naming of components. Read the following for a defense. But first, we generate this component using the following command:

### **Generate the `app_component` gem. Execute in ./**

[Click here to view code image](#)

```
$ rails plugin new components/app_component --full --mountable
```

---

## AppComponent Is Not a Good Component Name—Why Use It?

Let’s check `AppComponent` against the naming guidelines. First, it is specific and it does not conflict with common Rails terminology. So it passes the technical quality criteria. However, it obviously has no context-setting ability: It tells us only that it is the *component of an app*. Both of those things are obvious from the context; nothing is added by this name. `AppComponent` does not pass the most important naming quality criterion.

I am going to use `AppComponent` as the name for this component precisely *because* it has these properties! It is technically okay, and free of expectations regarding what it should (or does) contain. Because it is technically okay, we will be able to refactor things properly, and its ambiguity prevents us from getting stuck in any particular meaning of the name. This is important because we are about to dump an entire application into a single component. And while that is a good start, it is not

where we want to end up! We are going to be working with `AppComponent` throughout [Chapters 2](#) and [3](#). These chapters prepare us to finally, in Chapter 4, refactor away from this single (badly named) component toward multiple components with good names. Then, finally, all our components will adhere to all the naming guidelines laid out previously. Suffice it to say that at the very end of those refactorings, the small remaining piece of `AppComponent` will be renamed into `WebUI` and will contain only the aspects of the user interface shared by the entire application.

---

The command line parameters `--full` and `--mountable` make the plugin generator create a gem that loads a `Rails::Engine` that isolates itself from the Rails application. It comes with a test harness application and blueprints for tests based on `Test::Unit`. See Appendix A for a full explanation of these creation parameters and to learn how to switch the tests to RSpec.

We can now `cd` into the component folder in `./components/app_component` and execute `bundle`. Depending on our version of `bundler`, there will be a warning (in older versions) or error about our `gemspec` not being valid:

**app\_component gem error during bundle. Execute in  
./components/app\_component**

[Click here to view code image](#)

```
$ bundle
The latest bundler is 1.16.0.pre.3, but you are currently running
  1.15.4.
To update, run `gem install bundler --pre`
You have one or more invalid gemspecs that need to be fixed.
The gemspec at ./components/app_component/app_component.gemspec is
  not valid.
Please fix this gemspec.
The validation error was '"FIXME" or "TODO" is not a description'
```

This warning is because there are a bunch of “TODO” entries in the `gemspec` that are picked up on by `bundler` and that we need to remove. If you fill out all the fields with “TODO” in the `Gem::Specification` block of the `gemspec`, this warning/error will go away. However, email, homepage, description, and license are not required, and if you delete those you only need to fill in authors and summary to get rid of the warning.

Removing all TODOs from the generated `gemspec` leads to the following file, which will not throw an error:

**./components/app\_component/app\_component.gemspec**

[Click here to view code image](#)

```
1 $:.push File.expand_path("../lib", __FILE__)
2
3 # Maintain your gem's version:
4 require "app_component/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
```

```

8   s.name          = "app_component"
9   s.version       = AppComponent::VERSION
10  s.authors        = ["Stephan Hagemann"]
11  s.summary        = "Summary of AppComponent."
12  s.license         = "MIT"
13
14  s.files = Dir[ "{app,config,db,lib}/**/*",
15                  "MIT-LICENSE", "Rakefile", "README.md"]
16
17  s.add_dependency "rails", "~> 5.1.0"
18
19 end

```

If we now go back up into the root folder of Sportsball and call `bundle` there, we notice that among the many gems that are being used, we also see our new component `app_component`.

## **Seeing AppComponent being used in the app (some results omitted). Execute in ./**

[Click here to view code image](#)

```

$ bundle
The latest bundler is 1.16.0.pre.3, but you are currently running
  1.15.4.
To update, run `gem install bundler --pre`
Resolving dependencies...
Using rake 12.2.1

...
Using rails 5.1.4
Using sass-rails 5.0.6
Using app_component 0.1.0 from source at `components/app_component`
Bundle complete! 17 Gemfile dependencies, 73 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.

```

This is because `rails plugin new` not only creates a new gem, it also automatically adds a reference to this gem into the `Gemfile` of an app in the context of which it was created. That is, because we executed `rails plugin new` from the root of the Sportsball app, we got also got a new entry in our `Gemfile`.

## **./Gemfile showing reference to AppComponent gem (some lines omitted)**

[Click here to view code image](#)

```

1 source 'https://rubygems.org'
2
3 ...
4
5 gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
6 gem 'app_component', path: 'components/app_component'

```

In earlier versions of Rails, this does not happen automatically. If you are in this situation, add the last line from the previous snippet to your `Gemfile` to get the same result when bundling the app. Notice that the gem reference uses the `path` option to specify where this gem can be found (namely, our local filesystem). Commonly, `path` is used only for

gems under development, but as we will see, it works just fine for use in CBRA applications.

Back to AppComponent. While it is now hooked up within the Sportsball application, it does not actually do anything yet. Let us fix that next. We will create a landing page and route the root of the engine to it.

## Generate a controller for the component. Execute in ./

[Click here to view code image](#)

```
$ cd components/app_component  
$ rails g controller welcome index
```

By moving into the folder of the gem, we are using the engine's Rails setup. Because of that and because we made this a mountable engine, our welcome controller is created inside the AppComponent namespace. See Appendix B for an in-depth discussion of the folder structure and the namespace created by the engine.

We change the component's routes as follows to hook the new controller up to the root of the engine's routes:

**./components/app\_component/config/routes.rb**

[Click here to view code image](#)

```
1 AppComponent::Engine.routes.draw do  
2   root to: "welcome#index"  
3 end
```

Last, we make the main app aware of the routes of the engine by mounting them into the routes of the application. Here, we are mounting the engine to the root of the app as there are no other engines and the main app will never have any routes of its own. See Appendix B for a discussion of how routing works and what options you have.

**./config/routes.rb**

[Click here to view code image](#)

```
1 Rails.application.routes.draw do  
2   mount AppComponent::Engine, at: "/"  
3 end
```

That's all. Let's start up our server!

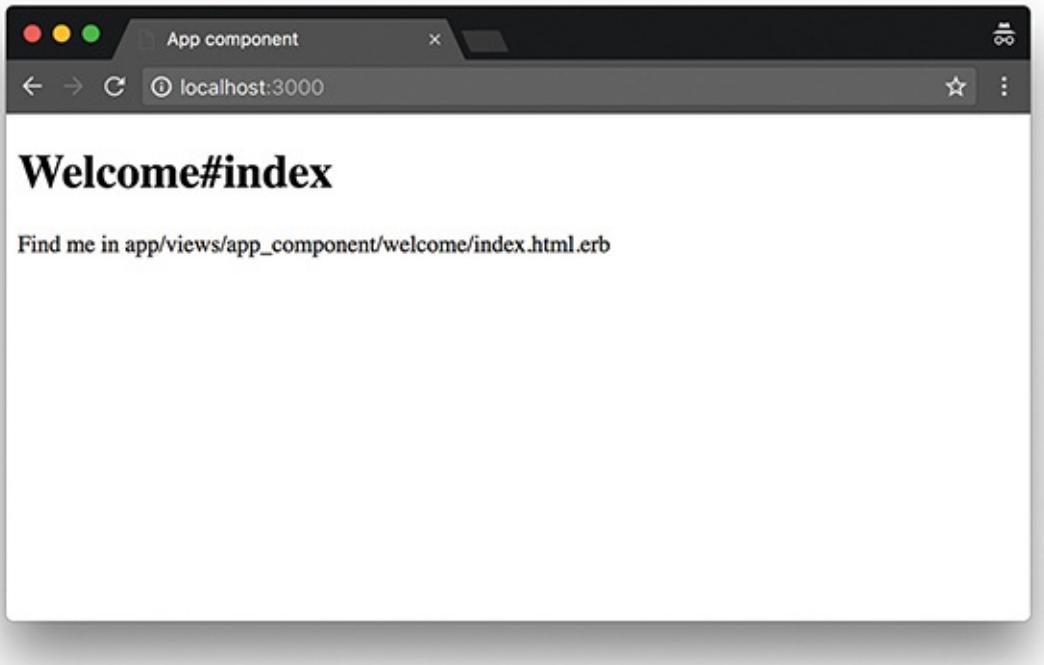
## Start up the Rails server. Execute in ./

[Click here to view code image](#)

```
$ rails s  
=> Booting Puma  
=> Rails 5.1.4 application starting in development  
=> Run `rails server -h` for more startup options  
Puma starting in single mode...  
* Version 3.10.0 (ruby 2.4.2-p198), codename: Russell's Teapot  
* Min threads: 5, max threads: 5  
* Environment: development  
* Listening on tcp://0.0.0.0:3000  
Use Ctrl-C to stop
```

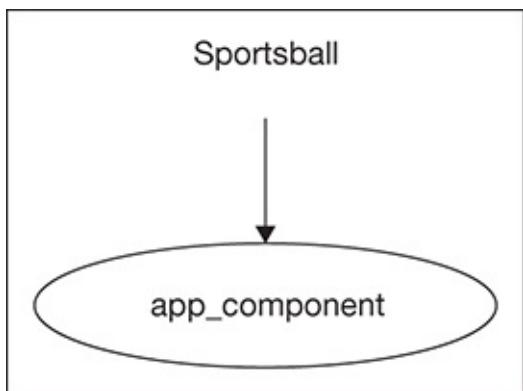
Now, when you open <http://localhost:3000> with a browser, you should see (as in [Figure](#)

[2.1](#)) your new welcome controller's index page. That wasn't too bad, was it?



**Figure 2.1.** Your first CBRA-run web page

Having the component separated from the container application also allows us to draw a component diagram (see [Figure 2.2](#)). This diagram shows our component in the middle using its gem name. The surrounding box with the application name indicates the Rails application of which our code is a part. The arrow indicates that the container app is dependent directly on `app_component` (as we just saw, `AppComponent` was added directly to the `Gemfile` of Sportsball). In subsequent sections of the book we will see how we create a web of components with dependencies among them.



**Figure 2.2.** Our first component diagram

Generating this graph yourself at this stage of the app is a bit tricky. It is the way we are referencing the component in `Gemfile` currently. If you still want to do it, there are a couple of steps involved.

First, you will need to install the `cobradeps` gem (<https://github.com/shageman/cobradeps>), a gem I wrote to allow for the generation of Rails component diagrams. This gem depends on `graphviz` (<http://www.graphviz.org/>), an open-source graph visualization software. You install these two applications as follows (assuming that you are on OSX and are `homebrew`, which I recommend you do):

<https://brew.sh/>).

## Install `graphviz` and `cobradeps`. Execute anywhere

[Click here to view code image](#)

```
$ brew install graphviz  
$ gem install cobradeps
```

To alleviate the gem reference problem we mentioned, change the `AppComponent` line in your `Gemfile` to this:

## Reference to `app_component` in `./Gemfile` allowing for graph generation

[Click here to view code image](#)

```
1 gem 'app_component',  
2   path: 'components/app_component',  
3   group: [:default, :direct]
```

The `group: [:default, :direct]` will behave normally with `bundler` and is used by `cobradeps` to determine that the gem is indeed a direct dependency (we will see in [Section 2.3](#) why this is necessary). Now you can generate the component graph by executing the following statement. The result will be output into `./component_diagram.png`.

## Generate a component graph for `Sportsball`. Execute in `./`

[Click here to view code image](#)

```
$ cobradeps -g component_diagram .
```

That's it! Your first CBRA app is ready to get serious. You can now continue all feature development inside of `components/app_component`.

Some aspects of what we glossed over in this section are covered in more depth in the Appendixes. Refer to Appendix A for an in-depth look at the various kinds of engines that Rails features, and Appendix B for an introduction to engine routing and mounting.

## 2.2 ActiveRecord and Handling Migrations within Components

Let's add some actual functionality to our currently bare-bones application. The first feature we are going to focus on is for the app to be able to predict the outcome of future games based on past performances. To this end, we will add teams and games as models to `AppComponent`. We will create an admin interface for both teams and games, which will give us enough data to try our hand at predicting some games.

Remember to execute these commands in the context of `AppComponent`, that is, under `./components/app_component`.

## Scaffolding Team and Game. Execute in `./components/app_component`

[Click here to view code image](#)

```
$ rails g scaffold team name:string
```

```
$ rails g scaffold game date:datetime \
    location:string \
    first_team_id:integer \
    second_team_id:integer \
    winning_team:integer \
    first_team_score:integer \
    second_team_score:integer
```

---

## Why I Continue to Use Scaffolds in Examples

When I use Rails in my job from day to day, I practically never use Rails' ability to generate scaffolds. There are many reasons for this: They are not what really needs to be built, they need to be adapted to work well, they are not driven by user stories, and they are not test-driven.

However, a book about large-scale patterns and architecture has to deal with a lot of code. Otherwise, it runs the risk of the patterns not being obvious or seeming nonsensical. For that, scaffolds are a great way to communicate about a lot of code without having to *show* a lot of code.

The next step is to run `rake db:migrate` to create the appropriate tables in the database. Interestingly, this will work when called within `./components/app_component`, but not in `./`. It does not fail for the main app. It just doesn't do anything.

### Scaffolding Team and Game. Execute in `./components/app_component`

[Click here to view code image](#)

```
$ rake db:migrate

== 20171029235211 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
 -> 0.0005s
== 20171029235211 CreateAppComponentTeams: migrated (0.0006s) =====

== 20171029235221 CreateAppComponentGames: migrating =====
-- create_table(:app_component_games)
 -> 0.0007s
== 20171029235221 CreateAppComponentGames: migrated (0.0007s) =====

$ cd ../..
$ rake db:migrate
Running via Spring preloader in process 58196
```

Before this will work, we need to make the main app aware of the migrations provided by the engine.

## 2.2.1 Installing Engine Migrations with Rake

The common solution to this is to install the engine's migrations into the main app with `rake app_component:install:migrations`. This will copy all the migrations found in the engine's `db/migrate` folder into that of the main app.

There are a few gems out there that use this functionality. For example, the gem `Commentator` (<https://github.com/lml/commentator>) does this. Most widely used gems, like `ActiveAdmin` (<http://activeadmin.info/>) and `Devise` (<http://devise.plataformatec.com.br/>), generate more complex migrations in the host app. They don't actually come with migrations of their own, but instead use generators to create them based on user-specified configuration options.

If we were to run `rake app_component:install:migrations` in the `Sportsball` app, we would get the following contents in the engine and the main app:

### Install engine migrations into the main app. Execute in `.`

[Click here to view code image](#)

```
$ rake app_component:install:migrations
Running via Spring preloader in process 58464
Copied migration 20171030000159_create_app_component_teams.\
 app_component.rb from app_component
Copied migration 20171030000160_create_app_component_games.\
 app_component.rb from app_component
```

### AppComponent engine migrations

[Click here to view code image](#)

```
$ tree ./components/app_component/db/migrate
components/app_component/db/migrate
└── 20171029235211_create_app_component_teams.rb
    └── 20171029235221_create_app_component_games.rb
```

### Sportsball application migrations

[Click here to view code image](#)

```
$ tree ./db/migrate
./db/migrate
└── 20170507205125_create_app_component_teams.app_component.rb
    └── 20170507205126_create_app_component_games.app_component.rb
```

While the original migrations in the engine are still present, the `rake` task has copied them into the main app. In doing so, it renamed them and changed their date to the current time. This ensures that the engine's migrations are being run as the last ones in the application ("last" at the time they are *installed* into the app).

The re-dating of migrations to the current time is very important for engines that are intended to be distributed (like the ones mentioned previously), because it is unknown when the gem, and thus its migrations, are going to be added to an application. If the dates were not changed, there would be no control over where in the overall migration sequence they would fall. However, since the gem had to have been published before the development of the app (or the relevant portion of the app) was started, they are likely going to be run very early. That, in turn, would likely lead to an invalid overall migration

state on any system that runs the app—even production: While newer migrations have run, older ones are missing.

Note that `rake railties:install:migrations` installs all new migrations from all engines in an application. If migrations are added after the installer has run, it will need to be run again to ensure all migrations are present.

Having to install migrations every time they are added to an engine is an extra step in comparison to what we are used to. And, as discussed previously, the reason it is needed in many situations (gems being developed independently from applications) does not hold true in our case. It turns out that we can change our engine to have the host Rails application automatically find and use its migrations.

## 2.2.2 Loading Engine Migrations in Place

Instead of copying migrations from components into the main application, we can ensure that the main app can find them with a few lines of code added to the component's `engine.rb`. This technique was first suggested by Ben Smith (<http://pivotallabs.com/leave-your-migrations-in-your-rails-engines/>).

**./components/app\_component/lib/app\_component/engine.rb – Engine migrations configuration**

[Click here to view code image](#)

```
1 module AppComponent
2   class Engine < ::Rails::Engine
3     isolate_namespace AppComponent
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11  end
```

Now `rake db:migrate` will find and correctly run the engine's migrations.

It is important not to use migration installation `rake` tasks in combination with this technique and to remove what may have been added while following the previous section. That would result in problems with migrations being loaded twice, which is similar to something we will see next.

## 2.2.3 Known Issues

There are a few snags you can run into with this approach that should be avoided.

### 2.2.3.1 Chaining db:migrate in Rake Tasks Fails to Add

For a still unknown reason, `rake db:drop db:create db:migrate` works fine for a normal Rails app, but fails to add the migrations when engine migrations are loaded in place. The simplest way around this is to split this command in two by running `rake db:drop db:create && rake db:migrate` instead. This has performance implications, as the `rake` now has to load twice.

Ben Smith proposes a few different fixes for this issue, the most concise being to require the Rails environment to be loaded before the `db:load_config` by adding the following file `db.rake`:

```
./components/app_component/lib/tasks/db.rake
```

[Click here to view code image](#)

```
1 Rake::Task["db:load_config"].enhance [:environment]
```

This has the side effect of the environment (i.e., the Rails app's contents) always being loaded before any database tasks are run. Not only does this have a performance implication, it also affects every environment. And while the pattern `rake db:drop db:create db:migrate` is common in development, it will never be called in the production environment. In my opinion, it is a bad tradeoff to affect production in unknown ways to achieve small benefits in development.

In conclusion, I recommend working around the problem instead of using a solution that is not fully understood. This issue should be fixed by understanding and fixing the true cause in whatever it is that Rails does to run migrations. That would also make a nice pull request to Rails for the one who finds it!

### 2.2.3.2 Other Rake Tasks Reported Not Working

It has been reported that other database-related tasks, like `rake db:setup`, `rake db:rollback`, and `rake db:migration:redo`, stop working with this approach. I have never been able to confirm this and it certainly works in the Sportsball app as we have created it so far.

### 2.2.3.3 Problems with Naming Engines

In the code snippet that adds the engine's migrations, you may have noticed the peculiar line `unless app.root.to_s.match root.to_s+File::SEPARATOR`. It prevents a problem with running `rake app_component:db:migrate` inside the engine itself, which throws the following error otherwise:

**Engine migrations loaded twice through dummy app. Executed in  
./components/app\_component (if the check is removed)**

[Click here to view code image](#)

```
$ rake app_component:db:migrate
rake aborted!
ActiveRecord::DuplicateMigrationNameError:
```

```
Multiple migrations have the name CreateAppComponentTeams
```

The match is a heuristic to determine whether the engine is currently being loaded within its own dummy app. If that is the case, the initializer should not add the routes, as they are added automatically. The heuristic fails if the dummy app of the engine is outside of its own directory structure, which should never be the case.

A less powerful version of this heuristic evaluates `app.root.to_s.match root.to_s`, which fails if used with two engines where the name of the including engine starts with the name of the included engine and they are in the same directory. For example, if `/components/users_server` depends on `/components/users`, the former matches the latter and will not load its migrations. The more robust version of the heuristic should be used to prevent this problem.

#### 2.2.3.4 Conclusion

We have added models for Team and Game and we have ensured that migrations can be run in the main application. Do so, if you haven't yet, and start the server.

**Run migrations and start the server. Executed in ./**

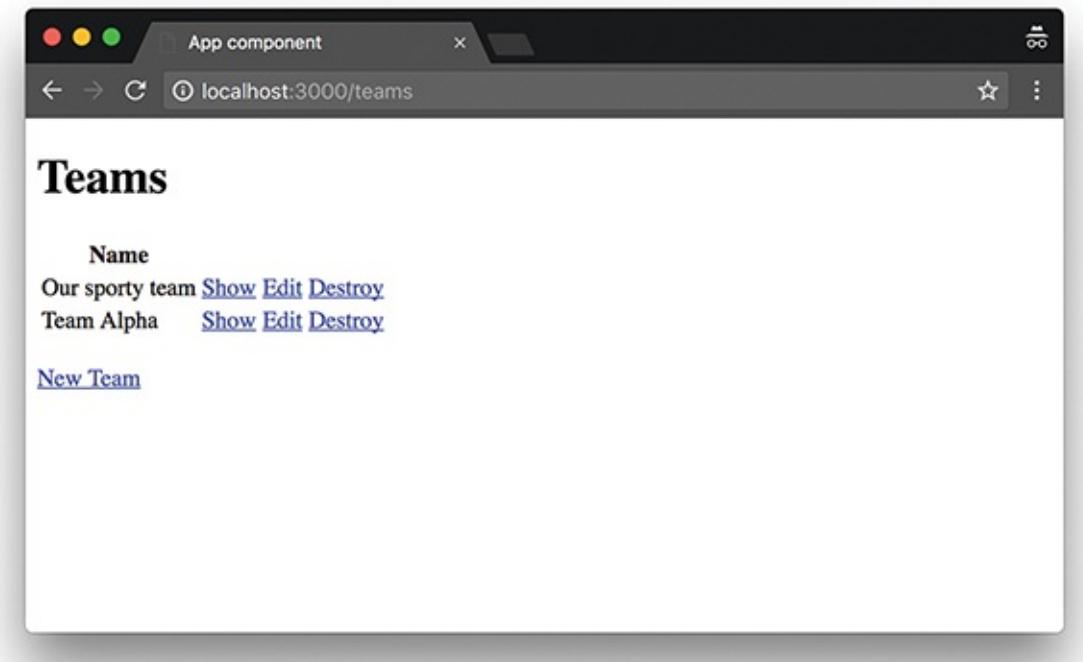
[Click here to view code image](#)

```
$ rake db:migrate
Running via Spring preloader in process 58740
== 20171029235211 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
 -> 0.0010s
== 20171029235211 CreateAppComponentTeams: migrated (0.0010s) =====

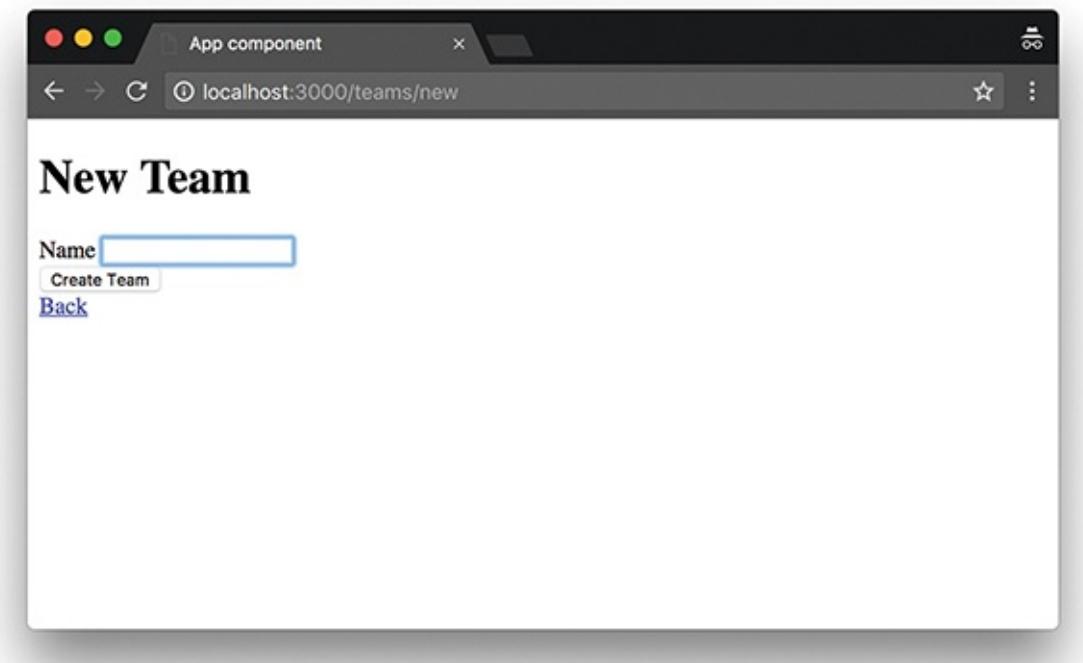
== 20171029235221 CreateAppComponentGames: migrating =====
-- create_table(:app_component_games)
 -> 0.0007s
== 20171029235221 CreateAppComponentGames: migrated (0.0007s) =====

$ rails s
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.10.0 (ruby 2.4.2-p198), codename: Russell's Teapot
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

With this, we can reach the UI for teams and games by navigating to <http://localhost:3000/teams> and <http://localhost:3000/games>, respectively. There, we are greeted with the standard look of scaffolded admin pages, as shown for teams in [Figures 2.3 and 2.4](#).



**Figure 2.3.** List of teams with two teams already added



**Figure 2.4.** Adding a new team

## 2.3 Handling Dependencies within Components

With Sportsball now having the ability to store teams and games, we can turn to the question of how to predict the outcome of games based on past performance. To this end, we would like to add a page that will allow us to pick two teams. Click a button labeled something like “Predict the winner!”, and see the application’s prediction of who is more likely to win as a result.

### 2.3.1 Using path Blocks for Specifying CBRA Dependencies

When we added our component, we used the following format for stating this dependency in the app's `Gemfile` as follows:

#### Sample `Gemfile` reference using path option

[Click here to view code image](#)

```
1 gem 'app_component', path: 'components/app_component'
```

There is another way of stating this dependency using a block syntax, like so:

#### Same `Gemfile` reference using path block

[Click here to view code image](#)

```
1 path "components" do
2   gem "app_component"
3 end
```

The first visible difference is that there will be less code to write when the list of dependencies grows. That is, of course, only if we put future components into the same components folder. Additional components are simply added to the block:

#### `Gemfile` reference with multiple gems in path block

[Click here to view code image](#)

```
1 path "components" do
2   gem "app_component"
3   gem "component_a"
4   gem "component_b"
5 end
```

There is another difference between the path option and the block syntax. As Enrico Teotti reports (<http://teotti.com/gemfiles-hierarchy-in-ruby-on-rails-component-based-architecture/>), the block syntax will use a feature in `bundler` that ensures that *transitive dependencies* of `AppComponent` are looked up in the stated path folder. That means that it will not be necessary to state every transitive CBRA dependency explicitly in the `Gemfile`. Instead, only the *direct* dependencies need to be listed.

For example, imagine that in the previous `Gemfile`, `component_a` depends on `component_c`. Without path block syntax, we would need to add `gem "component_c", path: "components/component_c"` to our `Gemfile`. With path block syntax, we don't have to. We get this for free since we already stated that the direct dependency `component_a` is listed.

Because of this, when using `cobradeps` to generate component diagrams, it is no longer necessary to specify a special group for direct dependencies; `cobradeps` simply assumes that all stated dependencies are direct dependencies.

### 2.3.2 Adding a Regular Gem: `slim`—Different Templating

Before we get to the part where we calculate a likely outcome, we need to add the new page. I find ERB unnecessarily verbose and avoid it when possible. Luckily, there are plenty of alternatives out there, and we can add the first dependency to our component that is not Rails.

I like `slim` (<http://slim-lang.com/>), as it greatly reduces the amount of code I have to write in comparison to ERB. Particularly, the number of chevrons (the “`<`” and “`>`” symbols so common in HTML) is greatly reduced, which I like a lot. Instead of adding the `slim` (<https://rubygems.org/gems/slim>) gem, we will add `slim-rails` (<https://rubygems.org/gems/slim-rails>), which in turn will require `slim`, but in addition adds Rails generators that can create views in `slim` syntax.

**`./components/app_component/app_component.gemspec` - Add slim dependency**

[Click here to view code image](#)

```
1 s.add_dependency "slim-rails"
```

This line should be added to `AppComponent`’s `gemspec` file to require `slim-rails`. Running `bundle` in the main app’s root folder, we should see `slim-rails` and `slim` being installed. Take a note of the exact version of `slim-rails` that is installed. At the time of writing, it is `3.1.3`.

To make use of our new gem, let us use the current welcome page as an example and translate it into `slim`. In fact, the current welcome page still contains that default auto-generated text. We will use the opportunity to give the page a bit more meaningful content.

So, let’s delete

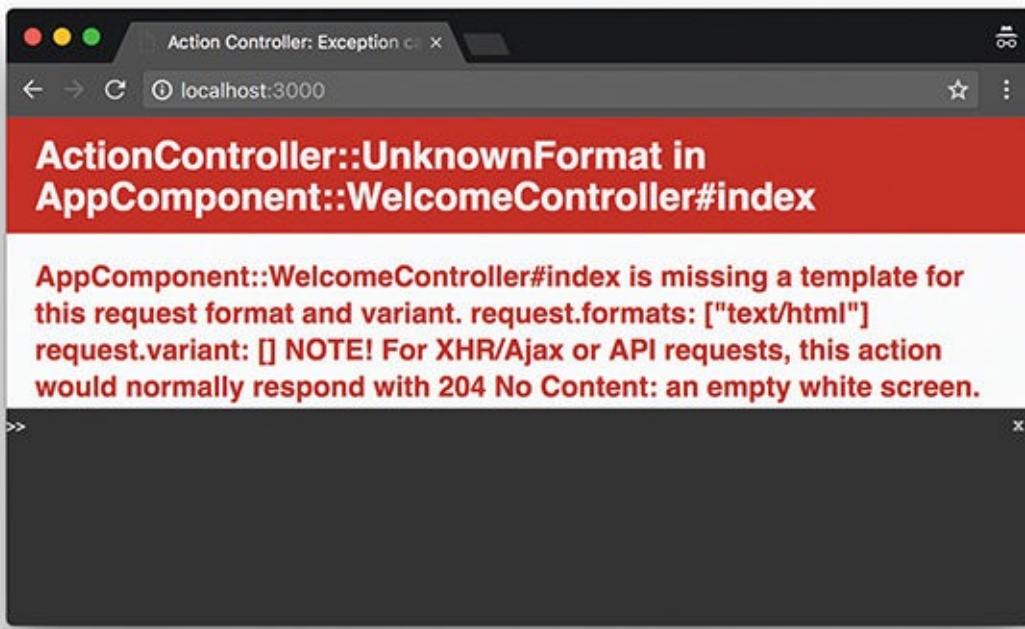
`./components/app_component/app/views/app_component/welcome/index.html.erb` and create an `index.html.slim` in the same folder instead. The new page links to the admin pages of Team and Game.

**`./components/app_component/app/views/app_component/welcome/index.html.slim`**

[Click here to view code image](#)

```
1 h1 Welcome to Sportsball!
2 p Predicting the outcome of matches since 2015.
3
4 = link_to "Manage Teams", teams_path
5 | &nbsp;| &nbsp;
6 = link_to "Manage Games", games_path
```

When we fire up the server, however, and try to load the new homepage of our app, instead of a page, we get this “Template is missing” error depicted in [Figure 2.5](#).



**Figure 2.5.** “Template missing” error after switching to `slim`

The reason for this is that, unlike Rails *applications*, which automatically require all the gems they are directly dependent upon, Rails *engines* do not. Check out Jonathan Rochkind’s blog post on the issue (<https://bibwild.wordpress.com/2013/02/27/gem-depends-on-rails-engine-gem-gotcha-need-explicit-require/>). We never require `slim` in our engine and it shows, because Rails reports only the template handlers that come standard: `:handlers=>[:erb, :builder, :raw, :ruby]`, but not `:slim` as we would expect.

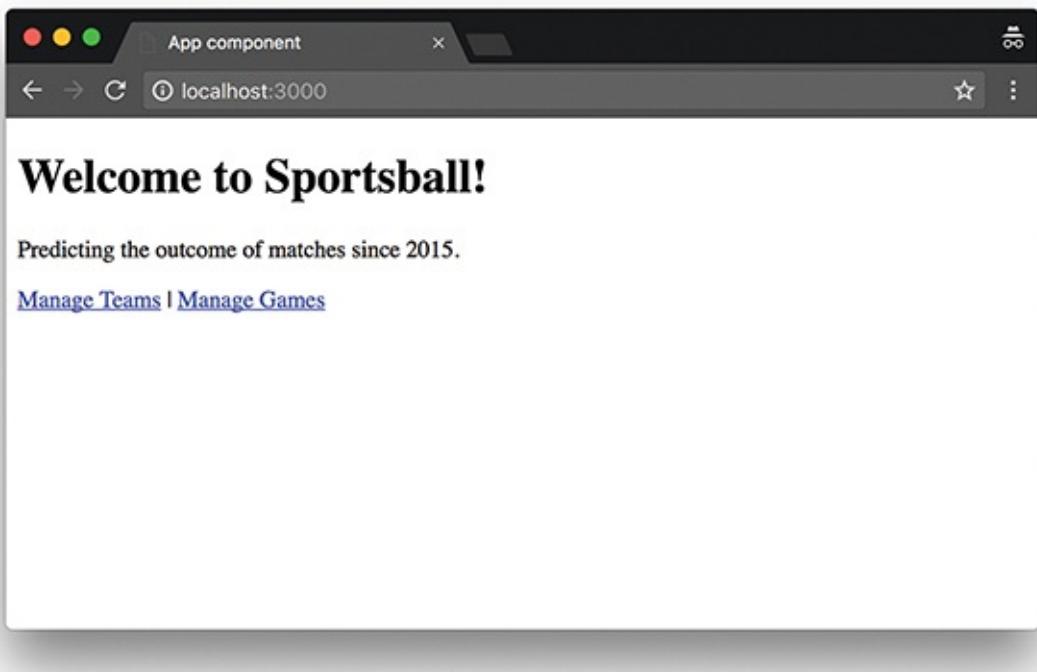
To fix the issue, we must explicitly require `slim-rails` in our `AppComponent` component, as follows. Note that I moved `require "app_component/engine"` into the scope of the `AppComponent` module. There is no programmatic need for that, but I like for gems to indicate this way which requires are *local* (i.e., within the gem) versus *external* (i.e., external gem dependencies).

**`./components/app_component/lib/app_component.rb - Require slim`**

[Click here to view code image](#)

```
1 require "slim-rails"
2
3 module AppComponent
4   require "app_component/engine"
5 end
```

We restart Rails to make it pick up the newly required gem and when we reload the homepage, we get the desired outcome, shown in [Figure 2.6](#).



**Figure 2.6.** New welcome page written in slim

## Making Custom Generators Work with Engines

We required `slim-rails` to be able to generate view code directly in `slim`. However, much like the gem itself, the availability of the generator itself is not sufficient for the component to start using it. If we were to repeat the generation of the Team and Game scaffolds from [Section 2.2](#), we would get `.erb` files once again.

In order for the engine's generators to pick `slim` as the templating library, the `AppComponent::Engine` generator definition needs to be configured as follows. Note that we can also set generators for object-relational mapping (ORM)—which is already `ActiveRecord` and which we don't want to change right now—and for the test framework, which we will get back to below. What is important right now is that the `template_engine` generator is switched to `:slim`. After this, all views will be generated using `slim` templates.

**./components/app\_component/lib/app\_component/engine.rb - Setting up the engine for different generators**

[Click here to view code image](#)

```
1 module AppComponent
2   class Engine < ::Rails::Engine
3     isolate_namespace AppComponent
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11  end
```

```
12 config.generators do |g|
13   g.orm           :active_record
14   g.template_engine :slim
15   g.test_framework  :rspec
16 end
17 end
18 end
```

---

### 2.3.3 Locking Down Gem Versions

Let us take another closer look at the runtime dependencies now present in our `AppComponent.gemspec`.

#### **`./components/app_component/app_component.gemspec - Production dependencies`**

[Click here to view code image](#)

```
1 s.add_dependency "rails", "~> 5.1.4"
2 s.add_dependency "slim-rails"
```

The Rails dependency was generated as `~> 5.1.4`, allowing all versions of Rails `5.1.*` (where `*` is 4 or greater).<sup>1</sup> We added `slim-rails` without any version restrictions.

Commonly, when developing gems, authors strive to keep the range of acceptable versions of needed gems as broad as possible. This is to exclude the fewest number of developers who might be in different situations and on different update paths from using a gem. Only for incompatible differences, which would prevent the gem from working properly, would a restriction typically be added.

Contrary to this, in Rails applications, `Gemfile.lock` is added to source control to lock down the versions of all dependencies. This ensures that when code is run in different environments or by different people, it will behave the same.

So. We are building an *app*, but are using *gems*. Which strategy should we take? Should we have a loose or a tight version policy? Well, I lock down all runtime dependencies in components to exact versions, like the following:

#### **`./components/app_component/app_component.gemspec - Production dependencies locked down`**

[Click here to view code image](#)

```
1 s.add_dependency "rails", "5.1.4"
2 s.add_dependency "slim-rails", "3.1.3"
```

The reason for the version lockdown has to do with the testing of the component and is based on a couple of assumptions. I assume that you write:

- Automated tests for your code
- Different kinds of tests, like unit, functional, integration, and feature
- Tests at the lowest possible level

If these assumptions are true for you, you will attempt to verify all of the internals of the

component within the component itself. That also means you will *not* be testing the internals *outside* of the component, that is, in the context of the completed Rails application. What would happen if, in this situation, the versions of dependencies of the component somehow drifted from the ones used in the Rails app? That would be an untested dependency in production: The functioning of the component would be verified against a version of its dependencies that are not used in production. The version lockdown enforces that all components bound together by the Rails app run with and are tested against the same version of a dependency. Testing components is the topic of the next section, [Section 3.1](#).

For this section, suffice it to say that there is merit to keeping the versions of dependencies in sync among all parts of the app. In the running app, only one version of every dependency is going to be loaded; we might as well try not to be surprised by how it works.

### 2.3.4 Adding the Development Version of a Gem: Trueskill—A Rating Calculation Library

We can now turn to the prediction of the outcome of future games. If we do not want to be in the business of figuring out how to do that, we better find a gem that will do such a calculation for us. Luckily, there is a lot of theory we could potentially draw from, such as ranking algorithms, rating algorithms, or Bayesian networks (<https://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html>). I started my search for a fitting gem with the FIFA World Rankings page on Wikipedia, which, while not explaining how the official rankings are calculated, mentions an alternative, the Elo rating system (The rating of chess players, past and present by Aspad Elo, 1978). Elo was created for use in chess but is now used in many competitor-versus-competitor games. An improvement to Elo is the Glicko rating system (<http://www.glicko.net/glicko.html>), which in turn was extended by Microsoft to TrueSkill (<http://trueskill.org/>), a system that works for multiplayer games. For all of these—Elo, Glicko, and Trueskill—we can find corresponding gems on rubygems (<https://rubygems.org>). For the following, we are going to work with the `trueskill` gem (<https://rubygems.org/gems/trueskill>). Not only does the idea of assessing a team’s strength while taking into account the players’ strengths sound appealing, but the gem also poses a nice little problem: It is totally outdated. At the time of writing, the last version of the gem was published in 2011. However, code has been contributed to forks of the original project until late 2014.

The version of the code we would like to use for `trueskill` is commit e404f45af5 (<https://github.com/benjaminleesmith/trueskill/tree/e404f45af5b3fb86982881ce064a9c764>) on the `benjaminleesmith` fork.<sup>2</sup> The problem is that we can only specify gems to depend on *published* versions of other gems. There is no way for us to set a restriction based on a commit SHA. For gems that are intended to be published, this makes sense: They should not depend on code that was not also published and distributed as a gem.

To work around this problem, we have to employ the gem’s `gemspec` and its `Gemfile` at the same time.

```
./components/app_component/Gemfile
```

[Click here to view code image](#)

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 gem "trueskill",
6   git: "https://github.com/benjaminleesmith/trueskill",
7   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

## **./components/app\_component/app\_component.gemspec – Dependencies**

[Click here to view code image](#)

```
1 s.add_dependency "rails", "5.1.4"
2 s.add_dependency "slim-rails", "3.1.3"
3 s.add_dependency "trueskill"
```

The `Gemfile` in a gem's directory is used during development of the gem, just like the `Gemfile` in a Rails app. When `bundle` is called in this directory, it will install all the gem dependencies listed there. The special line `gemspec` tells `bundler` to look for a `gemspec` file in the current directory and add all dependencies specified there to the current bundle. In our case, the `gemspec` states that `AppComponent` has a runtime dependency on `trueskill` and the `Gemfile` restricts this to be from the specified git URL at the given SHA.

**Bundle AppComponent with trueskill (some results omitted). Execute in  
./components/app\_component**

[Click here to view code image](#)

```
$ bundle
The latest bundler is 1.16.0.pre.3, but you are currently running \
  1.15.4.
To update, run `gem install bundler --pre`
Fetching https://github.com/benjaminleesmith/trueskill
Fetching gem metadata from https://rubygems.org/.....
Fetching version metadata from https://rubygems.org/...
Fetching dependency metadata from https://rubygems.org/..
Resolving dependencies...

...
Using trueskill 1.0.0 from https://github.com/benjaminleesmith/\
  trueskill (at e404f45@e404f45)

...
Bundle complete! 3 Gemfile dependencies, 46 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.
```

When bundling the component, we see git checking out the repository specified and using the correct SHA. However, bundling the main app will reveal that it does not take into account the restriction posed by the `Gemfile`. That is because the `Gemfile` of any gem is ignored by other gems or apps depending on it (again, due to fact that the common expectation is for a gem to be published).

To work around this, there is no other way than to ensure that the version of the dependency is enforced by the app itself. That leads to an exact duplicate of the `trueskill` line from `AppComponent`'s `Gemfile` in the main app's `Gemfile`.

## New lines in `./Gemfile`

[Click here to view code image](#)

```
1 gem "trueskill",
2   git: "https://github.com/benjaminleesmith/trueskill",
3   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

And just like with `slim-rails`, we need to explicitly require the `trueskill` gem in `AppComponent` to make sure it is loaded.

## `./components/app_component/lib/app_component.rb` - Requiring the trueskill dependency

[Click here to view code image](#)

```
1 require "slim-rails"
2 require "saulabs/trueskill"
3
4 module AppComponent
5   require "app_component/engine"
6 end
```

### 2.3.5 Adding Predictions to the App

With models, scaffolds for administration, and the rating calculation library in place, we can turn to implementing the first iteration of game prediction.

Let's create a cursory sketch of how our models might interact to generate a prediction. A `predictor` object might get a collection of all the games it should consider. As we are using an external library, we don't *really* know what is going on. The best way we can describe it is that the predictor *learns* (about the teams or the games). Because of this, we will make `learn` the first method of the public interface of the class.

After the predictor has learned the strengths of teams it can, given two teams, predict the outcome of their next match. `predict` becomes the second method of the public interface.

## `./components/app_component/app/models/app_component/predictor`

[Click here to view code image](#)

```
1 module AppComponent
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7           rating: [Saulabs::TrueSkill::Rating.new(
8             1500.0, 1000.0, 1.0)]
9         }
10        memo
11      end
```

```

12   end
13
14 def learn(games)
15   games.each do |game|
16     first_team_rating =
17       @teams_lookup[game.first_team_id][:rating]
18     second_team_rating =
19       @teams_lookup[game.second_team_id][:rating]
20     game_result = game.winning_team == 1 ?
21       [first_team_rating, second_team_rating] :
22       [second_team_rating, first_team_rating]
23     Saulabs::TrueSkill::FactorGraph.new(
24       game_result, [1, 2]).update_skills
25   end
26 end
27
28 def predict(first_team, second_team)
29   team1 = @teams_lookup[first_team.id][:team]
30   team2 = @teams_lookup[second_team.id][:team]
31   winner = higher_mean_team(first_team, second_team) ?
32     team1 : team2
33   AppComponent::Prediction.new(team1, team2, winner)
34 end
35
36 def higher_mean_team(first_team, second_team)
37   @teams_lookup[first_team.id][:rating].first.mean >
38     @teams_lookup[second_team.id][:rating].first.mean
39 end
40 end
41 end

```

To start, initialize creates a lookup hash from all the teams it is handed that allows the Predictor class to efficiently access teams and their ratings by a team's id.

Inside of learn, the predictor loops over all the games that were given. It looks up the ratings of the two teams playing each game. The teams' ratings are passed into an object from trueskill called FactorGraph in the order “winner first, loser second” so that the update\_skills method can update the ratings of both teams.

predict simply compares the mean rating values of the two teams and “predicts” that the stronger team will win. It returns a Prediction object, which we will look at next.

There is not much going on in the Prediction class. It is simply a data object that holds on to the teams participating in the prediction, as well as the winning team.

[./components/app\\_component/app/models/app\\_component/predictic](#)

[Click here to view code image](#)

```

1 module AppComponent
2   class Prediction
3     attr_reader :first_team, :second_team, :winner
4
5     def initialize(first_team, second_team, winner)
6       @first_team = first_team
7       @second_team = second_team

```

```

8     @winner = winner
9   end
10 end
11 end

```

The PredictionsController has two actions: new and create. The first, new, loads all teams so they are available for the selection of the game to be predicted. create creates a new Predictor and then calls learn and predict in sequence to generate a prediction.

[./components/app\\_component/app/controllers/app\\_component/predictions\\_controller.rb](#)

[Click here to view code image](#)

```

1 require_dependency "app_component/application_controller"
2 module AppComponent
3   class PredictionsController < ApplicationController
4     def new
5       @teams = AppComponent::Team.all
6     end
7
8     def create
9       predictor = Predictor.new(AppComponent::Team.all)
10      predictor.learn(AppComponent::Game.all)
11      @prediction = predictor.predict(
12        AppComponent::Team.find(params["first_team"]["id"]),
13        AppComponent::Team.find(params["second_team"]["id"]))
14    end
15  end
16 end

```

For completeness, we list the two views of the prediction interface as well as a helper that is used to generate the prediction result that will be displayed as a result.

[./components/app\\_component/app/views/app\\_component/predictions/new.html.slim](#)

[Click here to view code image](#)

```

1 h1 Predictions
2
3 = form_tag prediction_path, method: "post" do |f|
4   .field
5     = label_tag :first_team_id
6     = collection_select(:first_team, :id, @teams, :id, :name)
7
8   .field
9     = label_tag :second_team_id
10    = collection_select(:second_team, :id, @teams, :id, :name)
11   .actions = submit_tag "What is it going to be?", class: "button"

```

[./components/app\\_component/app/views/app\\_component/predictions/show.html.slim](#)

[Click here to view code image](#)

```

1 h1 Prediction
2
3 = prediction_text @prediction.first_team, @prediction.second_team,
  @prediction.winner

```

```

4
5 .actions
6   = link_to "Try again!", new_prediction_path, class: "button"
./components/app_component/app/helpers/app_component/predicti

```

[Click here to view code image](#)

```

1 module AppComponent
2   module PredictionsHelper
3     def prediction_text(team1, team2, winner)
4       "In the game between #{team1.name} and #{team2.name} " +
5       "the winner will be #{winner.name}"
6     end
7   end
8 end

```

Finally, we can add a link to the prediction to the homepage to complete this feature.

```

./components/app_component/app/views/app_component/welcome/
index.html.slim

```

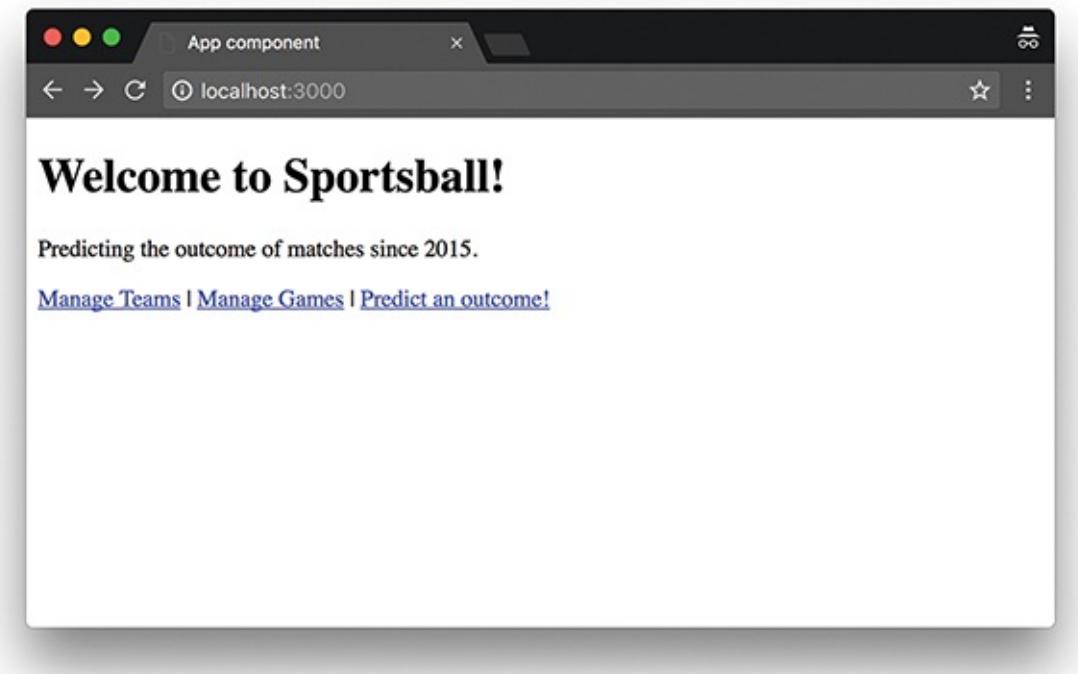
[Click here to view code image](#)

```

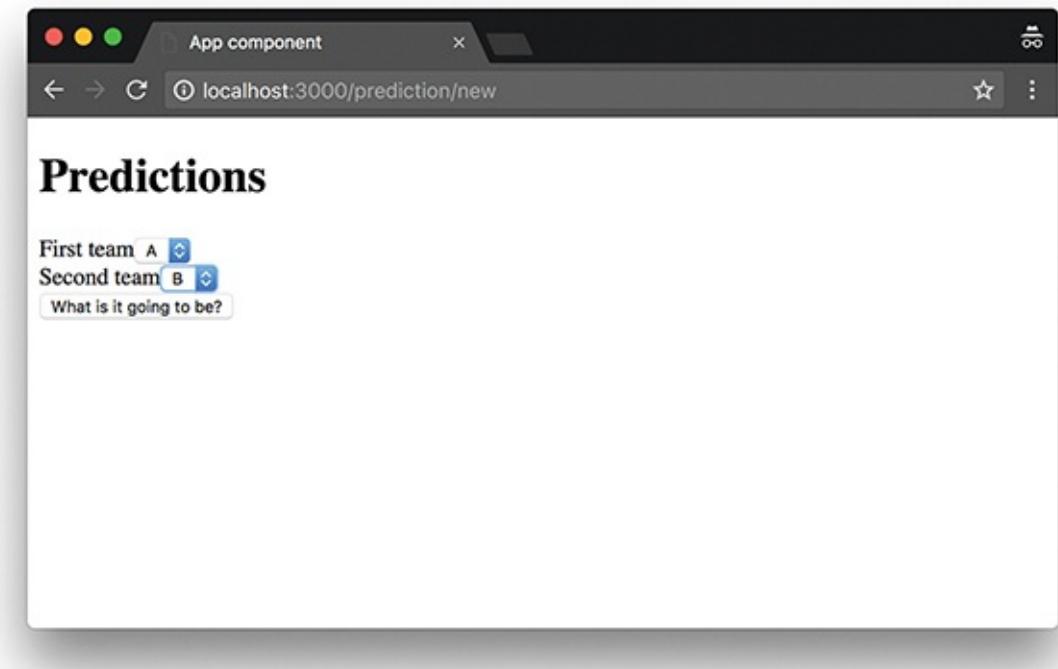
1 h1 Welcome to Sportsball!
2 p Predicting the outcome of matches since 2015.
3
4 = link_to "Manage Teams", teams_path
5 | &nbsp;| &nbsp;
6 = link_to "Manage Games", games_path
7 | &nbsp;| &nbsp;
8 = link_to "Predict an outcome!", new_prediction_path

```

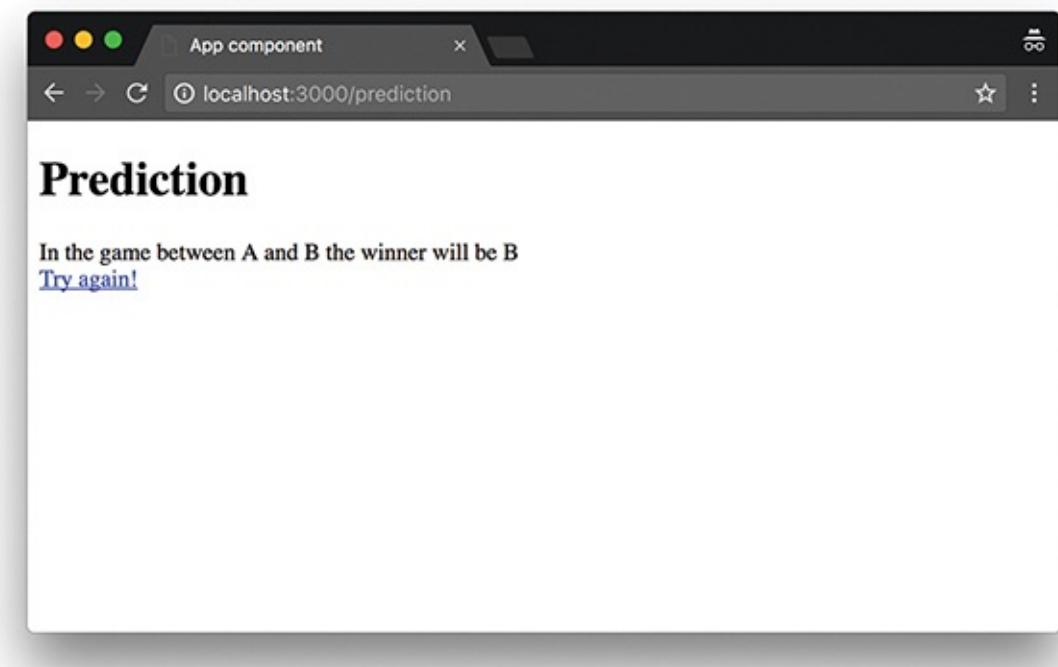
With the changes from this section in place, we can navigate to <http://localhost:3000/> to see a new homepage (see [Figure 2.7](#)) from which we can navigate to our new prediction section. Figure 2.8 shows how we can request a new prediction. Finally, in [Figure 2.9](#), we see the result of a successful prediction.



**Figure 2.7.** Sportsball homepage with link to predictions



**Figure 2.8.** Requesting the prediction of a game



**Figure 2.9.** Showing the prediction result

With the conclusion of this chapter, Sportsball is fully functional! Well, if you can call it functional at this point. In any case, the current state of the application will allow us to discuss and analyze many aspects of componentization. We will start in [Chapter 3](#) by testing our component and application.

1. For a full explanation of the possibilities of version restriction specifications, check out the Bundler page on gemfiles (<http://bundler.io/gemfile.html>) and the RubyGems Guide on patterns (<http://guides.rubygems.org/patterns/#declaring-dependencies>).
2. I know this fork works as intended, as I previously used it for the app that ranked foosball players in the Boulder Pivotal Labs office: true-foos-skills (<a href="https://github.com/shageman>true-foos-skills).

# CHAPTER 3

---

## Tooling for Components



Photo: francesco de marco/Shutterstock

With the essentials covered, we are still missing several tools and techniques to fully control a component-based application. That is what this chapter is going to cover. It explains how to test component-based applications, how to manage assets, how to manage dependencies long term, and how to deploy your application to production.

---

### Remember to Check Out the Book's Source Code

As was mentioned in [Chapter 2](#), from this chapter onward the text only discusses many, but not all changes to the source code. Please refer to the appropriate chapter folder in <https://github.com/shageman/component-based-rails-applications-book> to get the complete source code and a script to generate it all.

Most notably for this chapter, the entire Sportsball app was created again from scratch. This was to ensure that all the views were generated in `slim` and all the tests were generated using `rspec`. To do this, right after creating `AppComponent`,

```
./components/app_component/app_component.gemspec      and  
./components/app_component/lib/app_component/engine.rb to  
include all the needed dependencies and the generator config we discussed in  
Section 2.3.
```

---

## 3.1 Testing a Component

To test the component, we will add RSpec as our testing framework. Not only do I find RSpec a nice framework to use, it also provides us with an opportunity to discuss one more kind of dependency: development dependencies.

**`./components/app_component/app_component.gemspec - New rspec-rails reference`**

[Click here to view code image](#)

```
1 s.add_development_dependency "sqlite3"  
2 s.add_development_dependency "rspec-rails"
```

Remember that the dependency on sqlite3 was generated. We simply add rspec-rails in the same way. In contrast to the runtime dependencies we discussed in [Section 2.3](#), we do not have to lock down the version of these dependencies. It is acceptable for development dependencies to deviate between the component (or multiples) and the application, as they do not constitute production code and do not suffer from the problems we discussed before. However, development dependencies are not immune to test failures due to deviating dependency versions. If this is causing issues in your project, the answer is once again to lock down dependency versions or to at least get them into tighter bounds via optimistic or pessimistic version restrictions.

Recall the addition of prediction functionality in [Section 2.3](#): The new class Predictor has two main public methods, learn and predict. For predictor to work properly, Team and Game should always be non-nil and valid in the sense that teams have a name and games have two teams, scores, and a winner (or not). In fact, we want to enforce these aspects on the level of the models themselves so our data store never contains unusable data. Functionally, PredictionsController has to orchestrate these objects to trigger a prediction. We need a couple of views that allow a user to ask for and see a prediction.

To verify all of this, we need tests for ActiveRecord models, the Predictor class, and the feature as a whole. For the sake of completeness, we will also test the controllers, helpers, views, and routes.

---

### The Public Interface of a Class

It is not true that predictor has only two methods in its public interface. Simply by virtue of being of the class Predictor, which in turn is of the class Class, it has many methods. In fact, for Ruby version 2.1.5p273, Class adds 99 class methods and 55 instance methods to every subclass and their objects, respectively. As necessary as they are, the methods given to us by Ruby lack *intent* within our

application. That is the difference from the methods we create explicitly. So, when I write “public interface” of a class or object, I mean the “*intended* public interface” of that entity.

---

### 3.1.1 ActiveRecord Model Specs

The only methods we are going to be adding in Team and Game are from ActiveRecord. The `shoulda-matchers` (<https://rubygems.org/gems/shoulda-matchers>) gem offers a nice domain-specific language (DSL) for testing these. We add it to the development dependencies of our component and write the first test.

---

#### Not All `shoulda` Matchers Are Created Equal

`shoulda` has been criticized for not actually testing the behavior of an object, but merely its configuration. That is true for some, but not for all, of its matchers. `validates_presence_of` ([https://github.com/thoughtbot/shoulda-matchers/blob/948757b3f8af079869f2a94b628ee17db756b86d/lib/shoulda/matchers/active\\_record\\_matcher.rb#L11](https://github.com/thoughtbot/shoulda-matchers/blob/948757b3f8af079869f2a94b628ee17db756b86d/lib/shoulda/matchers/active_record_matcher.rb#L11)) for example, does check the fact that an attribute is present by trying to set it to nil and checking for an error. Association matchers like `belong_to` and `have_many` ([https://github.com/thoughtbot/shoulda-matchers/blob/948757b3f8af079869f2a94b628ee17db756b86d/lib/shoulda/matchers/association\\_matcher.rb#L11](https://github.com/thoughtbot/shoulda-matchers/blob/948757b3f8af079869f2a94b628ee17db756b86d/lib/shoulda/matchers/association_matcher.rb#L11)) however, don’t verify all of the behavior of the association; instead they only verify its configuration. Despite these shortcomings, we are going to use `shoulda` for all our model tests in this example.

Always make sure you understand your dependencies!

---

Let’s use Game as the exemplary ActiveRecord model spec. If you have used `shoulda` before, you will notice that nothing special is going on in the following test. Note that the `describe` is written for `AppComponent::Game`, which is due to every class inside of the `AppComponent` engine being namespaced with its module name.

`./components/app_component/spec/models/app_component/game_spec.rb`

[Click here to view code image](#)

```
1 RSpec.describe AppComponent::Game do
2   it { should validate_presence_of :date }
3   it { should validate_presence_of :location }
4   it { should validate_presence_of :first_team }
5   it { should validate_presence_of :second_team }
6   it { should validate_presence_of :winning_team }
7   it { should validate_presence_of :first_team_score }
8   it { should validate_presence_of :second_team_score }
9
10  it { should belong_to :first_team}
11  it { should belong_to :second_team}
12 end
```

Equally unsurprising as the test is the actual implementation of the Game class (repeated

in the following). The two relationships to the team class, `first_team` and `second_team`, need to be annotated with the classname of the relationships, as that cannot be inferred from the attribute name. Both `Team` and `AppComponent::Team` work for that.

[./components/app\\_component/app/models/app\\_component/game.rb](#)

[Click here to view code image](#)

```
1 module AppComponent
2   class Game < ApplicationRecord
3     validates :date, :location, :first_team, :second_team,
4               :winning_team, :first_team_score, :second_team_score,
5               presence: true
6     belongs_to :first_team, class_name: "Team"
7     belongs_to :second_team, class_name: "AppComponent::Team"
8   end
9 end
```

If executed at this point, the execution will fail with the following error message, because `rspec` has not been set up to work with the files of our gem.

**rspec unable to load files under test. Execute in**

[./components/app\\_component](#)

[Click here to view code image](#)

```
$ rspec spec
./components/app_component/spec/controllers/app_component/\
games_controller_spec.rb:1:in `<top (required)>':
  uninitialized constant AppComponent::GamesController (NameError)
```

For our tests to work, `spec_helper.rb` in the `spec` folder needs to be changed to load the files from the gem as follows:

[./components/app\\_component/spec/spec\\_helper.rb](#)

[Click here to view code image](#)

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10 require "ostruct"
11
12 require "rails-controller-testing"
13 Rails::Controller::Testing.install
14
15 Dir[AppComponent::Engine.root.join("spec/support/**/*.rb")].
16   each { |f| require f }
17
18 RSpec.configure do |config|
19   config.expect_with :rspec do |expectations|
20     expectations.
```

```

21     include_chain_clauses_in_custom_matcher_descriptions = true
22 end
23 config.mock_with :rspec do |mocks|
24   mocks.verify_partial_doubles = true
25 end
26
27 config.infer_spec_type_from_file_location!
28 config.disable_monkey_patching!
29 config.warnings = false
30 config.profile_examples = nil
31 config.order = :random
32 Kernel.strand config.seed
33
34 config.before(:suite) do
35   DatabaseCleaner.strategy = :transaction
36   DatabaseCleaner.clean_with(:truncation)
37 end
38
39 config.around(:each) do |example|
40   DatabaseCleaner.cleaning do
41     example.run
42   end
43 end
44
45 [:controller, :view, :request].each do |type|
46   config.include ::Rails::Controller::Testing::
47     TestProcess, :type => type
48   config.include ::Rails::Controller::Testing::
49     TemplateAssertions, :type => type
50   config.include ::Rails::Controller::Testing::
51     Integration, :type => type
52 end
53
54 config.include ObjectCreationMethods
55 end
56
57 Shoulda::Matchers.configure do |config|
58   config.integrate do |with|
59     with.test_framework :rspec
60     with.library :rails
61   end
62 end

```

In the preceding, line 3 loads the environment of the dummy app, which in turn loads all the files required within our gem. Lines 5 through 10 load external dependencies, which are needed to execute the specs. Not all these require statements are necessary for the game\_spec, but rather constitute everything that is necessary for all the tests in this section.

### 3.1.2 Non-ActiveRecord Model Specs

Next on the list is Predictor. Note first that I tend to place objects like Predictor in app/models despite their not inheriting from ActiveRecord::Base. Wherever classes like this are placed, testing them remains largely the same. The only thing one must look out for if placing classes outside of the standard paths is whether other paths still work with normal Rails autoloading or whether configurations must be set to allow for this.

Testing this class is really interesting from a test-writing perspective, as the class has deep functionality behind a relatively small public interface. From a component-testing perspective, however, testing predictor is the same as testing any other model. The tests attempt to capture the basic idea of the learning and predicting feature of the class without overspecifying how it is to be performed.

```
./components/app_component/spec/models/app_component/predictor_spec.rb
```

[Click here to view code image](#)

```
1 RSpec.describe AppComponent::Predictor do
2   before do
3     @team1 = create_team name: "A"
4     @team2 = create_team name: "B"
5
6     @predictor = AppComponent::Predictor.new([@team1, @team2])
7   end
8
9   it "predicts teams that have won in the past to win in the
10    future" do
11     game = create_game first_team: @team1,
12                         second_team: @team2, winning_team: 1
13     @predictor.learn([game])
14
15     prediction = @predictor.predict(@team2, @team1)
16     expect(prediction.winner).to eq @team1
17
18     prediction = @predictor.predict(@team1, @team2)
19     expect(prediction.winner).to eq @team1
20   end
21
22   it "changes predictions based on games learned" do
23     game1 = create_game first_team: @team1,
24                           second_team: @team2, winning_team: 1
25     game2 = create_game first_team: @team1,
26                           second_team: @team2, winning_team: 2
27     game3 = create_game first_team: @team1,
28                           second_team: @team2, winning_team: 2
29     @predictor.learn([game1, game2, game3])
30
31     prediction = @predictor.predict(@team1, @team2)
32     expect(prediction.winner).to eq @team2
33   end
34
35   it "behaves funny when teams are equally strong" do
36     prediction = @predictor.predict(@team1, @team2)
```

```

37   expect(prediction).to be_an AppComponent::Prediction
38   expect(prediction.first_team).to eq @team1
39   expect(prediction.second_team).to eq @team2
40   expect(prediction.winner).to eq @team2
41
42   prediction = @predictor.predict(@team2, @team1)
43   expect(prediction).to be_an AppComponent::Prediction
44   expect(prediction.first_team).to eq @team2
45   expect(prediction.second_team).to eq @team1
46   expect(prediction.winner).to eq @team1
47 end
48 end

```

The preceding tests are passed by the Predictor class repeated in the following. However, many usage (edge) cases are not covered by the tests or this implementation, such as the possibility of games being played between teams that were not previously loaded into the predictor. It does, however, do what we need it to do right now, and we can always add more complex behavior later.

The preceding test creates teams and games through the `create_team` and `create_game` methods. You can, of course, use your favorite way of creating objects in your tests. I have become used to writing the helper methods to do this myself, a process further described in the note titled “Object Creation in Tests.”

```
./components/app_component/app/models/app_component/predictor
```

[Click here to view code image](#)

```

1 module AppComponent
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7           rating: [
8             Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
9           ]
10        }
11        memo
12      end
13    end
14
15    def learn(games)
16      games.each do |game|
17        first_team_rating = @teams_lookup[game.first_team_id][
18          :rating]
19        second_team_rating = @teams_lookup[game.
20          second_team_id][:rating]
21        game_result = game.winning_team == 1 ?
22          [first_team_rating, second_team_rating] :
23          [second_team_rating, first_team_rating]
24        Saulabs::TrueSkill::FactorGraph.new(game_result, [1, 2])..
25          update_skills
26      end
27    end
28  end

```

```

29   def predict(first_team, second_team)
30     team1 = @teams_lookup[first_team.id] [:team]
31     team2 = @teams_lookup[second_team.id] [:team]
32     winner = higher_mean_team(first_team, second_team) ?
33       team1 : team2
34     AppComponent::Prediction.new(team1, team2, winner)
35   end
36
37   def higher_mean_team(first_team, second_team)
38     @teams_lookup[first_team.id] [:rating].first.mean >
39       @teams_lookup[second_team.id] [:rating].first.mean
40   end
41 end
42 end

```

---

## Object Creation in Tests

As [ruby-toolbox](https://www.ruby-toolbox.com/categories/rails_fixture_replacement) ([https://www.ruby-toolbox.com/categories/rails\\_fixture\\_replacement](https://www.ruby-toolbox.com/categories/rails_fixture_replacement)) shows, there are many libraries out there to replace fixtures for tests with some variation of the object mother pattern (<http://martinfowler.com/bliki/ObjectMother.html>). Like many others, I was in the habit of using those libraries until Jeff Dean reminded me (<http://pivotallabs.com/rolling-your-own-object-creation-methods-for-specs/>) what a simple pattern it is. Since then, I have written my own object creation methods, still very much in the same style. For the current iteration of Sportsball, my object creation methods look like this:

`./components/app_component/spec/support/object_creation_me`

[Click here to view code image](#)

```

1 module ObjectCreationMethods
2   def new_team(overrides = {})
3     defaults = {
4       name: "Some name #{counter}"
5     }
6     AppComponent::Team.new {
7       |team| apply(team, defaults, overrides)
8     }
9   end
10
11  def create_team(overrides = {})
12    new_team(overrides).tap(&:save!)
13  end
14
15  def new_game(overrides = {})
16    defaults = {
17      first_team: -> { new_team },
18      second_team: -> { new_team },
19      winning_team: 2,
20      first_team_score: 2,
21      second_team_score: 3,
22      location: "Somewhere",
23      date: Date.today
24    }

```

```

25
26     AppComponent::Game.new {
27         |game| apply(game, defaults, overrides)
28     }
29 end
30
31 def create_game(overrides = {})
32     new_game(overrides).tap(&:save!)
33 end
34
35 private
36
37 def counter
38     @counter ||= 0
39     @counter += 1
40 end
41
42 def apply(object, defaults, overrides)
43     options = defaults.merge(overrides)
44     options.each do |method, value_or_proc|
45         object.__send__((
46             "#{method}=",
47             value_or_proc.is_a?(Proc) ?
48                 value_or_proc.call : value_or_proc)
49     end
50 end
51 end

```

---

While Predictor itself does not inherit from or depend on ActiveRecord, its specs require it so that the parameters to learn and predict can be created. If one digs deeper, one will find that that is not true and that the specs can be written without any ActiveRecord being involved. We will get back to this point in [Chapter 4](#) where this dependency will pose a nice refactoring challenge.

### 3.1.3 Controller Specs

The prediction part of the app is also what we turn to for an example of controller tests. The following is a first pass at a test for PredictionController. In addition to what we previously discussed, we created a Prediction class, which is a value object that holds on to the competing teams and the winning team.

`./components/app_component/spec/controllers/app_component/pre`

[Click here to view code image](#)

```

1 RSpec.describe AppComponent::PredictionsController,
2     :type => :controller do
3     routes { AppComponent::Engine.routes }
4
5     before do
6         @team1 = create_team
7         @team2 = create_team
8     end
9
10    describe "GET new" do

```

```

11     it "assigns all teams as @teams" do
12       get :new, params: {}, session: {}
13       expect(assigns(:teams)).to eq [@team1, @team2]
14     end
15   end
16
17   describe "POST create" do
18     it "assigns a prediction as @prediction" do
19       post :create,
20         params: {
21           first_team: {id: @team1.id},
22           second_team: {id: @team2.id}
23         },
24         session: {}
25
26
27       prediction = assigns(:prediction)
28       expect(prediction).to be_a AppComponent::Prediction
29       expect(prediction.first_team).to eq @team1
30       expect(prediction.second_team).to eq @team2
31     end
32   end
33 end

```

The implementation of the controller we already saw in [Section 2.3](#) makes the tests pass.

[./components/app\\_component/app/controllers/app\\_component/predictions\\_controller\\_spec.rb](#)

[Click here to view code image](#)

```

1 require_dependency "app_component/application_controller"
2 module AppComponent
3   class PredictionsController < ApplicationController
4     def new
5       @teams = AppComponent::Team.all
6     end
7
8     def create
9       predictor = Predictor.new(AppComponent::Team.all)
10      predictor.learn(AppComponent::Game.all)
11      @prediction = predictor.predict(
12        AppComponent::Team.find(params["first_team"]["id"]),
13        AppComponent::Team.find(params["second_team"]["id"]))
14    end
15  end
16 end

```

Note the `routes { AppComponent::Engine.routes }` in line 3 of the spec. This ensures that, for this example group, RSpec uses the routes as defined by the engine and not those defined by the dummy app. Without this addition and in contrast to testing controllers in Rails apps, these tests would not work out of the box. For example, the error one would get for the spec of “GET new” is:

**rspec failing due to unknown routes (some results omitted). Execute in**

[./components/app\\_component](#)

[Click here to view code image](#)

\$ rspec spec

```
Failure/Error: get :new, {}, {}
ActionController::UrlGenerationError:
  No route matches {:action=>"new",
  :controller=>"app_component/predictions"}
```

RSpec's routes method is not supported in Rails 3.<sup>1</sup> If running on Rails 3, the issue can be fixed by adding `use_route: :app_component` (where `app_component` is the snake-case<sup>2</sup> name of the engine) as a parameter to every `get`, `post`, `put`, or `delete` call in a spec. In the given case, that looks like so: `get :new, {use_route: :app_component}, {}`.

As that becomes cumbersome fast, an often-suggested solution is to override the default implementation of those methods as follows. Ensure this file is loaded from `spec_helper.rb` to write tests exactly the same as for Rails apps.

## **./components/app\_component/spec/support/controller\_spec\_fixes** **Fix for Rails 3 route loading problem**

[Click here to view code image](#)

```
1 module EngineControllerTestMonkeyPatch
2   def get(action, parameters = nil, session = nil, flash = nil)
3     process_action(action, parameters, session, flash, "GET")
4   end
5
6   def post(action, parameters = nil, session = nil, flash = nil)
7     process_action(action, parameters, session, flash, "POST")
8   end
9
10  def put(action, parameters = nil, session = nil, flash = nil)
11    process_action(action, parameters, session, flash, "PUT")
12  end
13
14  def delete(action, parameters = nil, session = nil, flash = nil)
15    process_action(action, parameters, session, flash, "DELETE")
16  end
17
18  private
19
20  def process_action(action, parameters = nil, session = nil,
21                     flash = nil, method = "GET")
22    parameters ||= {}
23    process(action,
24            method,
25            parameters.merge!({:use_route => :app_component},
26            # :app_component == engine name
27            session,
28            flash)
29  end
30 end
31
32 AppComponent::Engine.routes.
```

```

33     default_url_options = {host: "test.host"}
34 EngineRoutes = AppComponent::Engine.routes.url_helpers
35
36 RSpec.configure do |config|
37   config.include EngineControllerTestMonkeyPatch,
38     :type => :controller
39 end

```

Without lines 32–34, the engine’s routes are not available to tests. These lines ensure that routes are configured and accessible in tests in the EngineRoutes namespace. For example, predictions\_path would be called as EngineRoutes.predictions\_path.

If possible, you should upgrade Rails to a version that supports RSpec’s routes to circumvent all these hacks.

### 3.1.4 Helper Specs

Helper specs are exactly the same as they are for a Rails app. I commonly avoid having any helpers, but still created a short one for the sake of completeness: PredictionHelper.prediction\_text expects to be passed three parameters that each have a name method. From these, it constructs the string that is used to report the outcome of a prediction.

[./components/app\\_component/spec/helpers/app\\_component/predict](#)

[Click here to view code image](#)

```

1 require "spec_helper"
2
3 RSpec.describe AppComponent::PredictionsHelper, :type => :helper do
4   it "returns a nice prediction text" do
5     Named = Struct.new(:name)
6     text = prediction_text(
7       Named.new("A"), Named.new("B"), Named.new("C"))
8     expect(text).to eq
9     "In the game between A and B the winner will be C"
10  end
11 end

```

[./components/app\\_component/app/helpers/app\\_component/predict](#)

[Click here to view code image](#)

```

1 module AppComponent
2   module PredictionsHelper
3     def prediction_text(team1, team2, winner)
4       "In the game between #{team1.name} and #{team2.name} " +
5       "the winner will be #{winner.name}"
6     end
7   end
8 end

```

Since the test and implementation themselves don’t offer much topic for discussion, let us turn to a more general discussion of how to test namespaced classes.

As we have seen a couple of times, the one thing that keeps repeating is the

`AppComponent::` prefix, which is needed to denote that we are in the component's namespace. One can get around having to repeat the `AppComponent::`, which fully qualifies the name of the object under test, by ensuring that the entire test is in the lexical scope of the `AppComponent` module.<sup>3</sup> For the preceding test, that would look as follows:

### **`predictions_helper_spec.rb` with `AppComponent` as module scope**

[Click here to view code image](#)

```
1 module AppComponent
2   RSpec.describe PredictionsHelper, :type => :helper do
3     it "returns a nice prediction text" do
4       Named = Struct.new(:name)
5       text = prediction_text(
6         Named.new("A"), Named.new("B"), Named.new("C"))
7       expect(text).to eq
8         "In the game between A and B the winner will be C"
9     end
10   end
11 end
```

In this particular case, we don't gain much, but for more complex classes, this is tempting —it might remove the need for quite a few `AppComponent::` constructs! The interesting side effect is that that following class would in fact pass the test we now have:

**`./components/app_component/app/helpers/app_component/predictions_helper.rb` that passes the new test. Bad.**

[Click here to view code image](#)

```
1 module PredictionsHelper
2   def prediction_text(team1, team2, winner)
3     "In the game between #{team1.name} and #{team2.name} " +
4       "the winner will be #{winner.name}"
5   end
6 end
```

Note that this helper module is not in the `AppComponent` namespace! The reason this still works is that Ruby searches for every constant in the current lexical scope of namespaces first. Then, successively removing the last element of the scope (until it is empty), it checks the parent namespaces. The only way to prevent this is to re-add the `AppComponent::` prefix to the test.

There are merits to both ways of doing it, and even to combining the two. In the end, it comes down to whether we can still be confident in the tests we have written.

### 3.1.5 Feature Specs

To add feature specs, I typically add capybara (<http://teamcapybara.github.io/capybara/>) as a gem dependency. With regard to testing a component, the only thing of note is the fact that the first line of the actual test (line 17) visits '/app\_component/'. This is because, as discussed in Appendix B, the dummy app mounts the engine under test at its snake-cased name by default, which is app\_component in our case.

```
./components/app_component/spec/features/predictions_spec.rb
```

[Click here to view code image](#)

```
1 require "spec_helper"
2
3 RSpec.describe "the prediction process", :type => :feature do
4   before :each do
5     team1 = create_team name: "UofL"
6     team2 = create_team name: "UK"
7
8     create_game first_team: team1,
9                   second_team: team2, winning_team: 1
10    create_game first_team: team2,
11                   second_team: team1, winning_team: 2
12    create_game first_team: team2,
13                   second_team: team1, winning_team: 2
14  end
15
16  it "get a new prediction" do
17    visit "/app_component/"
18
19    click_link "Predictions"
20
21    select "UofL", from: "First team"
22    select "UK", from: "Second team"
23    click_button "What is it going to be"
24
25    expect(page).to have_content "the winner will be UofL"
26  end
27 end
```

---

## How I Actually Write My Tests

In this section, I have listed all available types of specs. Half of these I never use in practice.

I have never written *routing* and *view* specs (when they still existed, before Rails 5). I avoid writing *helpers* (and their specs) and instead create presenters or similar concepts that I place next to the models they present in the app\_component/models folder. I would like controllers to always be so simple that I never have to write *controller* specs. However, I do tend to write them and, when I do, I try to use them to create good interactions with models and services.

I write model specs, controller specs, and feature specs. *Features* I write to ensure

that the overall functionality works—as such, they tend to go full stack.

---

## 3.2 Testing the Main Application

With the singular component in our application tested, the focus can turn back to the main application and the question: Is there anything on this side of the house that needs testing?

The only pieces of code that we have changed so far are `routes.rb` and `Gemfile`. We should have changed `README`, but haven't done so yet; it would also not affect any testing. Every other piece of code is still as Rails generated it. That is, every piece of code that was not deleted. The `app` folder was the first thing to go. In the current version of the code, `lib` and `vendor` have also been deleted.

As discussed before, I believe that code should be tested at the lowest level possible so that the tests are as fast and as isolated as they can be. Here, there is hardly any code; it is all boilerplate and the only two changes are about gluing the component into the Rails app. To me, it follows that the only thing in need of testing is the fact that this “gluing” worked. And for this, I like to use a feature spec.

### 3.2.1 What to Test in the Main App

The following tests verify four aspects of the application: that the main app is mounted, that the teams page loads, that the games page loads, and that a prediction can be performed.

`./spec/features/app_spec.rb`

[Click here to view code image](#)

```
1 require "spec_helper"
2
3 RSpec.describe "the app", :type => :feature do
4   it "hooks up to /" do
5     visit "/"
6     within "main h1" do
7       expect(page).to have_content "Sportsball"
8     end
9   end
10
11  it "has teams" do
12    visit "/"
13    click_link "Teams"
14    within "main h1" do
15      expect(page).to have_content "Teams"
16    end
17  end
18
19  it "has games" do
20    visit "/"
21    click_link "Games"
22    within "main h1" do
23      expect(page).to have_content "Games"
24    end
```

```

25   end
26
27 it "can predict" do
28   AppComponent::Team.create! name: "UofL"
29   AppComponent::Team.create! name: "UK"
30
31   visit "/"
32   click_link "Predictions"
33   click_button "What is it going to be"
34 end
35 end

```

The following spec verifies the correct loading and mounting of the AppComponent component as succinctly as possible. First (line 5) it visits /, the root of the application, which, as set in [Section 2.1](#), is where the component is mounted. This contrasts with the feature spec for the prediction functionality in [Section 3.1](#), which visited /app\_component because of how the dummy app mounts the component. Then, in line 6, the spec verifies a single piece of content, namely the header “Sportsball”, to be present on the visited page. This ensures not only that the content can be found, but in effect also that there are no errors on the page, and that the component is where we expect it.

### 3.2.2 How to Put Everything Together

With the test for the main application, there are now two test suites one can run: cd sportsball && rspec spec and cd sportsball/components/app\_component && rspec spec, and both need to run in order to verify that everything in the app works. While two commands instead of one may at first not seem like a big deal, we should strive to keep the simplicity that is embodied in “one command to run all tests.”

I tend to implement this functionality with shell scripts. And the solution I am going to propose here is to create one shell script per component and an additional one as a runner for all components.

**./components/app\_component/test.sh**

[Click here to view code image](#)

```

1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running app component engine specs"
6 bundle
7 bundle exec rake db:create db:migrate
8 RAILS_ENV=test bundle exec rake db:create db:migrate
9 bundle exec rspec spec
10 exit_code+=$?
11
12 exit $exit_code

```

**./test.sh**

[Click here to view code image](#)

```

1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running container app specs"
6 bundle
7 bundle exec rake db:drop
8 bundle exec rake environment db:create db:migrate
9 RAILS_ENV=test bundle exec rake environment db:create db:migrate
10 bundle exec rspec spec
11 exit_code+=$?
12
13 exit $exit_code

```

At this stage, the two `test.sh` files are essentially the same. For convenience, they both include bundling as well as creating and migrating of the database into one script together with the execution of the actual test. Only if every step in the script passes does the test script report a zero exit status to denote a successful execution.

In the future, the scripts may grow to be different—for example, if one of them needs to execute JavaScript tests or if the feature specs are broken out from the other specs. The only property they really need to keep is that they return a zero exit code if everything went well and a positive one if something went wrong.

To allow the joint execution of these tests scripts, we create the third script as `build.sh` in the root of the application.

### **`./build.sh`**

[Click here to view code image](#)

```

1 #!/bin/bash
2
3 result=0
4
5 cd "$( dirname "${BASH_SOURCE[0]}" )"
6
7 for test_script in $(find . -name test.sh); do
8   pushd `dirname $test_script` > /dev/null
9   ./test.sh
10  ((result+=${?}))
11  popd > /dev/null
12 done
13
14 if [ $result -eq 0 ]; then
15   echo "SUCCESS"
16 else
17   echo "FAILURE"
18 fi
19
20 exit $result

```

`build.sh` exploits the fact that the test scripts for the main app and all its components are called `test.sh` and are present in the subfolders of the build script itself. The loop in lines 7 through 12 finds all test scripts, switches into their respective folders, executes the tests, and tallies up the results. If any of the test scripts return a positive exit code, the exit

code of the build script will also be positive and thus indicate a build failure. As the following listing shows, the different test suites' outputs are reported one after the other, and any failures can easily be traced back to a particular part of the test suite in a particular component.

**Run `./build.sh` to see all tests pass! Execute in `./`**

[Click here to view code image](#)

```
$ ./build.sh
*** Running app component engine specs
.....
```

```
Finished in 0.91921 seconds (files took 1.65 seconds to load)
70 examples, 0 failures
```

```
Randomized with seed 6563
```

```
*** Running container app specs
```

```
db/test.sqlite3 already exists
```

```
.
```

```
Finished in 0.07369 seconds (files took 1.56 seconds to load)
1 example, 0 failures
```

```
Randomized with seed 29486
```

```
SUCCESS
```

Alternatively, the functionality `test.sh` and `build.sh` can be implemented through Ruby scripts. The reasons I use shell scripts instead are problems with nested system calls to Ruby and weird situations with unexpected exit codes. By using shell as we have shown, the calculation of a successful execution will not have those problems.

### 3.2.3 How to Set Up a Continuous Integration (CI) Server

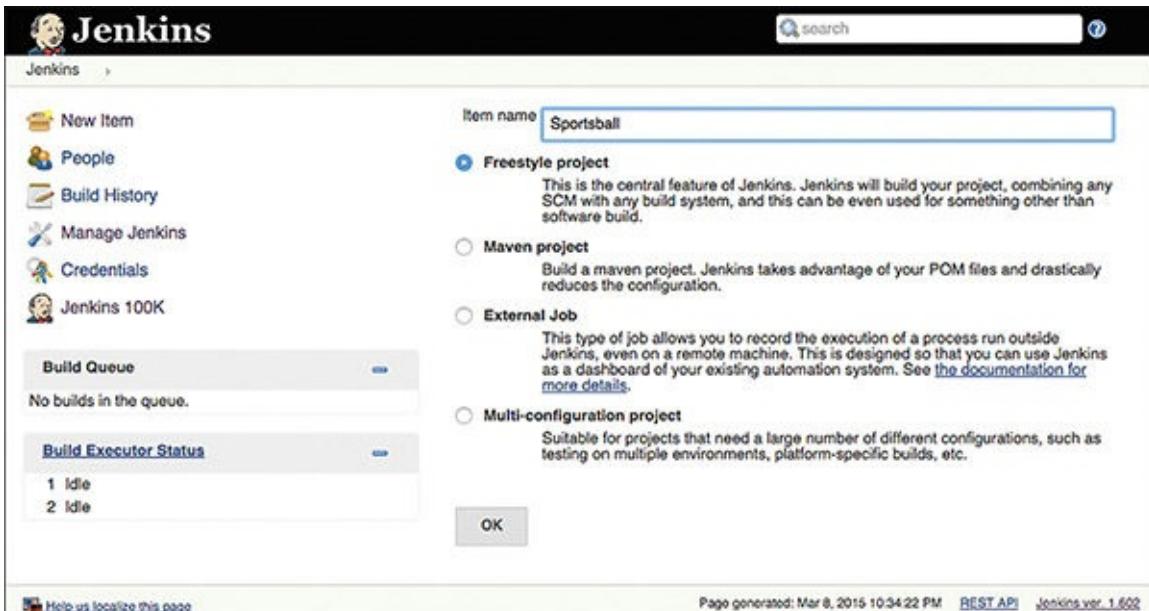
We will discuss two examples of CI servers in this section. The first, Jenkins, is a popular example of a self-hosted continuous integration server. The second, Travis CI, is one of the popular Software-as-a-Service (SaaS) continuous integration services.

#### 3.2.3.1 Setting Up Your Jenkins

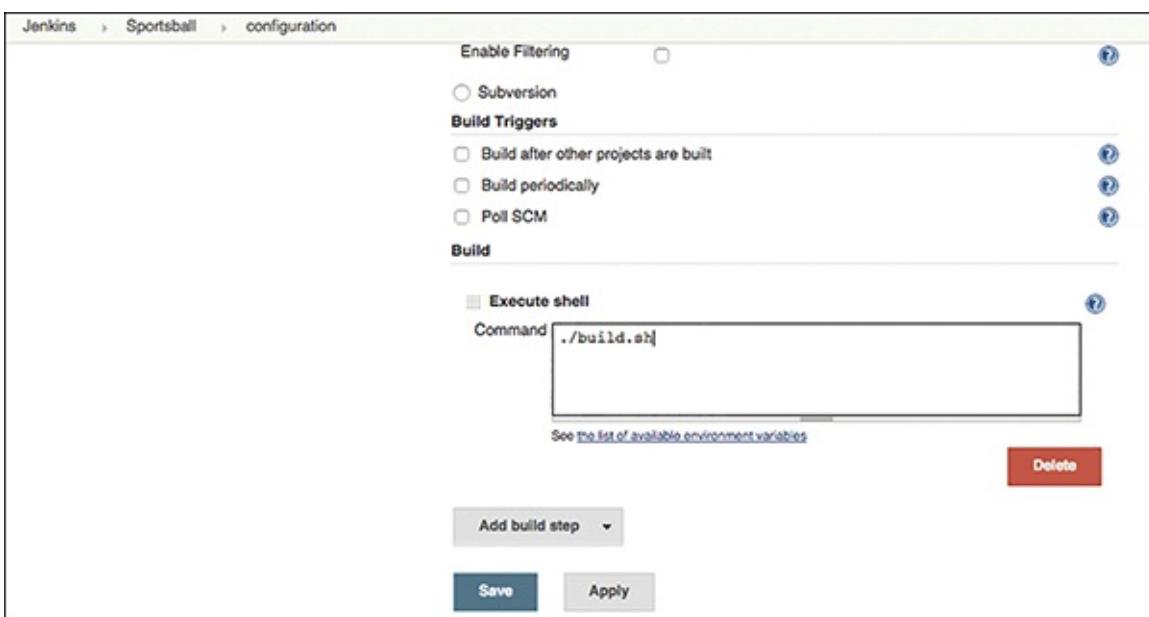
If we assume that your Jenkins (<http://jenkins-ci.org/>) server is already set up, there is not much to do to make it build Sportsball.

On the Jenkins home screen, click “New Item” to create a new build (Figure 3.1). Select

the “Freestyle project” option and give the new test suite a good name. Set the appropriate source within the “Source Code Management” section. Finally, within the “Build” section, click “Add Build Step” and choose to add “Execute shell”. The “Command” field should only take one line as configuration (Figure 3.2): `./build.sh`.



**Figure 3.1.** Adding new build to Jenkins



**Figure 3.2.** Jenkins build setup

Depending on how well your Jenkins server is set up, this is all there is to it. If something fails, check the log of the build to determine what is missing.<sup>4</sup> Problems could be caused by Ruby not being installed, the bundler gem not being installed, insufficient permissions for installing gems, or the installation of a required gem failing on a native extension. In all these cases, follow any instructions given in the error message to fix the problem.

In case you are using RVM to manage your Rubies, read the RVM guide on Jenkins (<http://rvm.io/integration/jenkins>).

On my home machine, I use “Jenkins in your Dock!” to run Jenkins and I use RVM

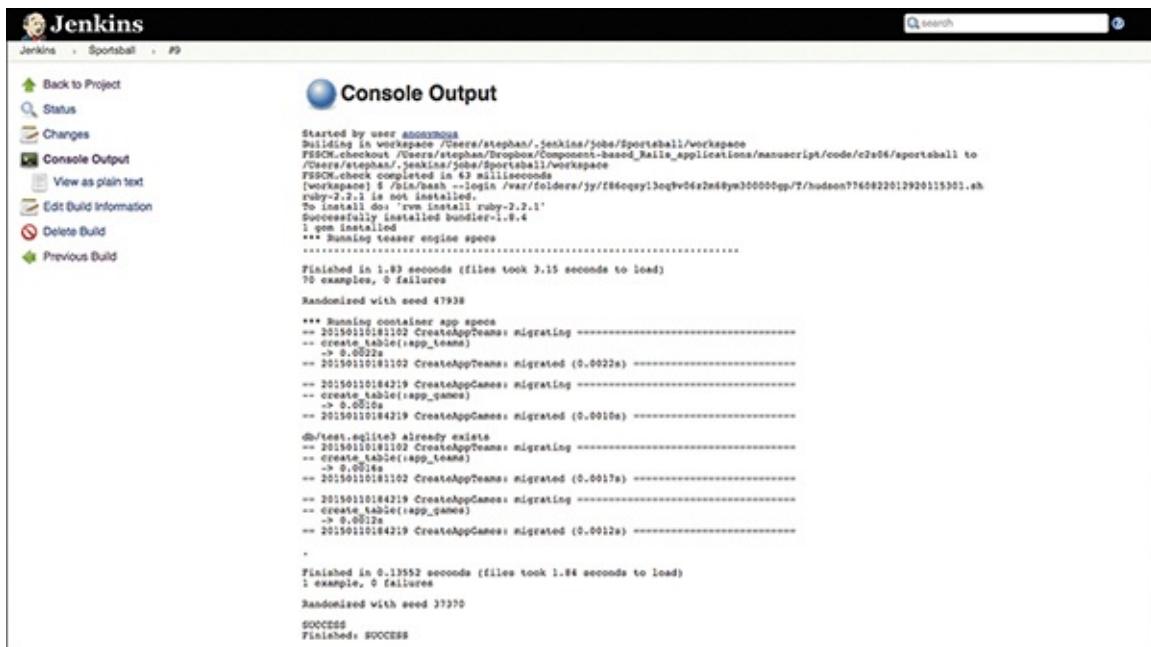
(<https://github.com/stisti/jenkins-app>). For this setup, the build command needs to be changed to the following for the build to succeed:

## Sufficient ‘Execute shell’ command

[Click here to view code image](#)

```
1 #!/bin/bash --login
2 export PATH="$PATH:$HOME/.rvm/bin" # Add RVM to PATH for scripting
3 rvm use "2.4.2@sportsball"
4
5 gem install bundler
6 ./build.sh
```

Figure 3.3 shows the output of a successful build.



The screenshot shows the Jenkins interface for a 'Sportsball' job. On the left, there's a sidebar with links like 'Back to Project', 'Status', 'Changes', 'Console Output' (which is selected), 'View as plain text', 'Delete Build', and 'Previous Build'. The main area is titled 'Console Output' and displays the terminal logs of the build process. The logs show the build starting, cloning the repository, installing dependencies, running migrations, and finally concluding with a 'SUCCESS' status.

```
Started by user anonymous
Building in workspace /Users/stephen/.jenkins/jobs/Sportsball/workspace
FSSCM.checkout /Users/stephen/Dropbox/Component-based_Rails_applications/manuscript/code/c2a06/sportsball
FSSCM.checkout completed in 43 milliseconds
[workspace] $ /bin/bash --login /var/folders/jy/f86eqy13oq9v06x2n68ym300000gp/T/hudson7765822012920115301.sh
ruby-2.7.1 is not installed.
To install, run 'gem install ruby-2.7.1'
Successfully installed bundler-1.9.4
1 gem installed
*** Running testee engine specs
-----
Randomized with seed 47938
*** Running container app specs
-- 201901101812102 CreateAppTeams: migrating -----
-- create_table(:app_teams)
--> 0.0022s
-- 201901101812102 CreateAppTeams: migrated (0.0022s) -----
-- 20190110184219 CreateAppGames: migrating -----
-- create_table(:app_games)
--> 0.0010s
-- 20190110184219 CreateAppGames: migrated (0.0010s) -----
db/test.sqlite3 already exists
-- 201901101812102 CreateAppTeams: migrating -----
-- create_table(:app_teams)
--> 0.0016s
-- 201901101812102 CreateAppTeams: migrated (0.0017s) -----
-- 20190110184219 CreateAppGames: migrating -----
-- create_table(:app_games)
--> 0.0012s
-- 20190110184219 CreateAppGames: migrated (0.0012s) -----
.
.
.
Finished in 0.13952 seconds (files took 1.84 seconds to load)
1 example, 0 failures
Randomized with seed 37370
SUCCESS
Finished: SUCCESS
```

Figure 3.3. Successful Jenkins build

### 3.2.3.2 Setting up Travis CI

As an example of how to set up a CBRA build with one of the many hosted CI solutions, we will also set up a build on Travis CI (<https://travis-ci.org>). The choice is warranted by the service’s current popularity and by its approach, namely the configuration of builds through the `.travis.yml`, which has been picked up by other CI services as the way to configure builds (<http://docs.travis-ci.com/user/build-configuration/>).

A first attempt could be to replicate the Jenkins configuration as closely as possible, which would lead to the following `.travis.yml` file:

**`./.travis.yml - naive version`**

[Click here to view code image](#)

```
1 rvm:
2   - 2.4.2
3 script: ./build.sh
```

This configuration will not lead to a successful build. The output of this configuration for the Sportsball app is available via the Travis CI logs at

<https://s3.amazonaws.com/archive.travis-ci.org/jobs/52271428/log.txt>. The following is the error that fails the build:

## Build failure with naive `.travis.yml` file. Push your code: Travis CI will execute this for you

[Click here to view code image](#)

```
*** Running app component engine specs
...
/gems/activesupport-4.1.9/lib/active_support/dependencies.rb:247:in \
`require': cannot load such file -- shoulda/matchers (LoadError)
```

The problem with this configuration is that Travis reads only one `Gemfile` per build run and the calls inside the `test.sh` scripts to bundle additional gemsets are unsuccessful.

One way of fixing this is for the root `Gemfile` to contain all the gems needed by any of the components of the application. This way, the main application's `Gemfile` contains a superset of all the sets of gems needed by the components of the app and the test suites of all components can find all the gems they depend on.

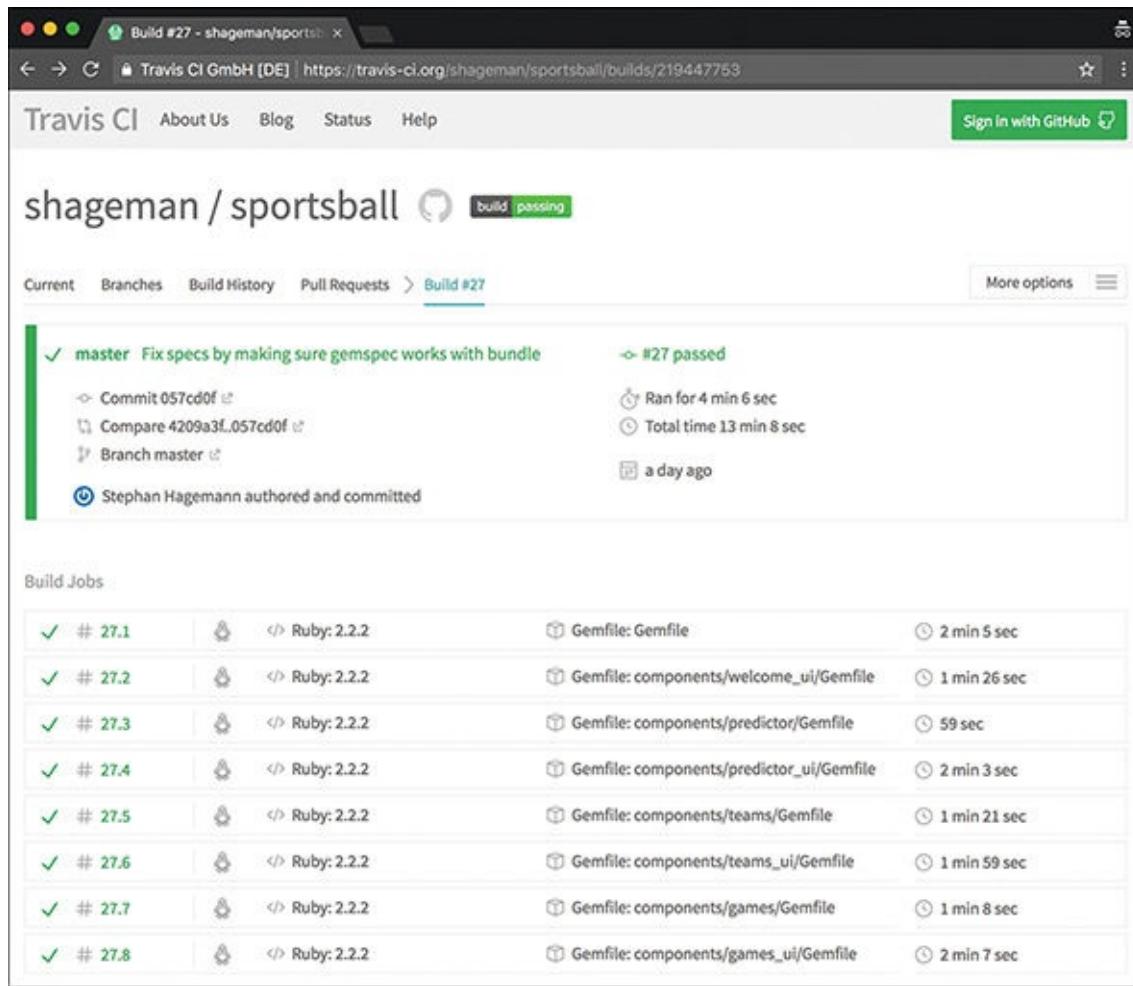
There is another way to fix the issue, however—one that translates what `build.sh` is trying to achieve into the way Travis runs things. This pattern is courtesy of Enrico Teotti, who first showed it for `the_next_big_thing` ([https://github.com/shageman/the\\_next\\_big\\_thing/commit/3b835359e8d64a65a75ea1c13f4](https://github.com/shageman/the_next_big_thing/commit/3b835359e8d64a65a75ea1c13f4))

## `./.travis.yml` - multiple test suites and retry

[Click here to view code image](#)

```
1 gemfile:
2   - Gemfile
3   - components/app_component/Gemfile
4 script:
5   - travis_retry ./test.sh
6 before_install:
7   - cd $(dirname $BUNDLE_GEMFILE)
8 rvm:
9   - 2.4.2
```

The first change to note is the presence of `Gemfile`. Normally used to create build matrices that test different versions of dependencies by loading different gemfiles, here it is used to reference the gemfiles of the main app and those of all the components. In combination with `before_install`, this trick achieves the intended effect: Before running the test script, as defined in `script`, Travis changes the directory to the location of the `Gemfile`. This creates, in effect, a build matrix of all the components of the application. Again, there is log output of the successful run at <https://travis-ci.org/shageman/sportsball/builds/52273098> on Travis CI. On the web site, the build output looks as shown in [Figure 3.4](#).



**Figure 3.4.** Successful Travis build (Screenshot of Travis © Travis CI, GmbH)

The `travis_retry` in line 5 is a fix for network timeouts that sometimes fail Travis builds. As “On Network Timeouts and Build Installation Retries” (<https://blog.travis-ci.com/2013-05-20-network-timeouts-build-retries/>) describes, the use of `travis_retry` leads our custom build commands to be tried up to three times before the build actually fails.

### 3.3 Asset Loading for Components

While the application is functionally complete for its first iteration, we have not spent much energy on making the views look more appealing. That is what we are going to tackle in this section. More broadly, we are going to discuss how to configure assets on a per-component basis so we achieve “asset independence” by default. This is a necessary precursor to setting up clean asset sharing in the future.

Sportsball is going to use the Foundation Responsive Library (<http://foundation.zurb.com>) as the basis for its front-end structure, layout, and style. I went to the Foundation web page and downloaded a customized version of the library for which I picked a few custom colors. The resulting zip file contains a bunch of files; next to the Foundation styles and JavaScript files, there are its dependencies, both CSS and JS.

To get the `AppComponent` component to pick up all files necessary for Foundation, I recreated the gem’s `vendor` folder and added all the library’s files as shown in the following. One file from the Foundation package does, in fact, not have to be installed,

namely jquery, as that comes standard with Rails.

## Folder structure under `./components/app_component/vendor`

[Click here to view code image](#)

```
./components/app_component/vendor
└── assets
    ├── javascripts
    │   └── app_component
    │       ├── fastclick.js
    │       ├── foundation.min.js
    │       ├── jquery.cookie.js
    │       ├── modernizr.js
    │       └── placeholder.js
    └── stylesheets
        └── app_component
            ├── foundation.min.css
            └── normalize.css
```

For AppComponent to pick up these new files, we need to add them to its `application.js` and `application.css` files. As load order is important for these library files, every single dependency is required explicitly and not via `require_tree`.

## application.js and application.css—Careful with Other Names

If you do not name your components' JS and CSS files `application.{js|css}`, you are going to run into issues. See the following for how to address those!

For `application.js`, there are two things of note. First, `jquery` (line 2) and `jquery_ujs` (line 7) are required without the `app_component/` prefix, as they are loaded from Rails' default location. Second, lines 9 through 11 contain the JavaScript code block needed to initialize Foundation's JavaScript parts after the web page is finished loading and the DOM is ready.

`./components/app_component/app/assets/javascripts/app_compone`

[Click here to view code image](#)

```
1 // = require app_component/modernizr.js
2 // = require jquery
3 // = require app_component/jquery.cookie.js
4 // = require app_component/placeholder.js
5 // = require app_component/fastclick.js
6 // = require app_component/foundation.min.js
7 // = require jquery_ujs
8
9 $(document).ready(function() {
10     $(document).foundation();
11});
```

`./components/app_component/app/assets/stylesheets/app_compone`

[Click here to view code image](#)

```
1 /*  
2 *= require app_component/normalize  
3 *= require app_component/foundation.min  
4 *= require_self  
5 */
```

The preceding sets up JavaScript and stylesheets to be loaded as `app_component/application.js` and `app_component/application.css`. Consequently, we see these locations show up in the layout of the `AppComponent` component in lines 9 and 10.

[./components/app\\_component/app/views/layouts/app\\_component/app\\_component/app\\_component.html.haml](#)

[Click here to view code image](#)

```
1 |<!DOCTYPE html>  
2 html  
3   head  
4     meta charset="utf-8"  
5     meta name="viewport" content="width=device-width,  
       initial-scale=1.0"  
6  
7   title Sportsball App  
8  
9   = javascript_include_tag "app_component/application"  
10  = stylesheet_link_tag    "app_component/application",  
     media: "all"  
11  
12  = csrf_meta_tags  
13  
14 header  
15   .contain-to-grid.sticky  
16     nav.top-bar data-topbar="" role="navigation"  
17       ul.title-area  
18         li.name  
19           h1  
20             = link_to root_path do  
21               = image_tag "app_component/logo.png", width: "25px"  
22               | Predictor  
23  
24         li.toggle-topbar.menu-icon  
25           a href="#"  
26             span Menu  
27  
28       section.top-bar-section  
29         ul.left  
30           li =link_to "Teams", teams_path  
31           li =link_to "Games", games_path  
32           li =link_to "Predictions", new_prediction_path  
33  
34 main  
35   .row  
36     .small-12.columns  
37       = yield
```

To complete the tour of assets, a logo image was added to the page (see line 21 in the

preceding). It resides in `./sportsball/components/app_component/app/assets/images/app_component/logo.png`. Thankfully, only the last part of this long path is needed to load the image, and in the view it can be referred to as `app_component/logo.png`.

---

## Why Not Use the foundation-rails Gem

If you are using a front-end library that has any traction in the community, chances are there is a gem named `somelibrary-rails` or `rails-somelibrary` that promises to handle asset loading for us. There is such a gem for Foundation—`foundation-rails` (<https://rubygems.org/gems/foundation-rails>)—but I recommend that you do not use it. If you *do* try to use it, be warned that you might see weird behavior.

There are a couple of reasons that I do not use them for components:

- These libraries use `Rails::Engine` themselves. Engines used within engines often require special setup to ensure they work properly, as their built-in expectation is that they are used directly within a Rails application.
- Gems can only be loaded once, which forces all components to use the same version of the library. This will become relevant and will require more discussion when we create applications with multiple components, especially multiple front-end components.
- With multiple components, there can be unexpected load order conflicts that result in the libraries being picked up in one but not all components. Again, this will be discussed in future chapters.

Foundation has many more than one gem to its name on rubygems (<https://rubygems.org/search?utf8=%E2%9C%93&query=foundation>). Be cautious with front-end gems.

---

### 3.3.1 Production-Ready Assets

As we have indicated, issues arise with assets when the base of the name of the main file is something other than `application`.

In development, this problem does not surface, as Rails serves assets in a debug mode by default (check `config/environment/development.rb` for `config.assets.debug = true`). In this mode, Rails does not attempt to compile its JS and CSS assets into a single, optimized file and instead will look for all assets in the source tree of the application. Because of this, production can be broken even though development is fine. Essentially what is happening is that Rails does not compile assets named in a non-standard way by default.

### 3.3.1.1 How to Check Whether Production Is Okay

This technique is useful for verifying that assets are fine in production, but can be used more broadly to verify that the application behaves as expected in production environment. To begin, a couple of settings need to be adjusted:

- Export a temporary secret into the shell the Rails server is going to run in by running

[Click here to view code image](#)

```
export SECRET_KEY_BASE=something_to_test
```

This assumes that you have not changed the default config/secrets.yml—you can temporarily set a secret there too.

- Change config/environments/production.rb to ensure that config.serve\_static\_assets = true
- Set the production database to a local test database in config/database.yml
- Create the database and run migrations for the production environment:

[Click here to view code image](#)

```
RAILS_ENV=production rake db:create  
RAILS_ENV=production rake db:migrate
```

- Start the server in the production environment with

```
RAILS_ENV=production rails s
```

Visiting the page without any further actions will result in a totally broken web page, as production assets are expect to be precompiled and Rails will not attempt to locate any missing files in the source tree of the application.

rake assets:precompile will perform the compilation necessary for assets to work in production.

#### Precompile assets. Execute in . /

[Click here to view code image](#)

```
$ RAILS_ENV=production rake assets:precompile  
I, [2015-02-22T20:45:47.062189 #55331] INFO -- : Writing  
./sportsball/public/assets/app_component/\  
logo-14e0571820ed8569b86519648f035ddd.png  
I, [2015-02-22T20:45:49.695917 #55331] INFO -- : Writing  
./sportsball/public/assets/app_component/\  
application-6e039fbe90e5f75b1b3a88842d1face8.js  
I, [2015-02-22T20:45:50.829624 #55331] INFO -- : Writing  
./sportsball/public/assets/app_component/\  
application-097b9ea4ccfc13ae1d38051c047dc0d.css
```

The task writes the resulting assets into public/assets. Disregarding the long hash present in the generated filename, it becomes clear which source files were compiled—in the preceding case, logo.png, application.js, and application.css.

### 3.3.1.2 Allowing for Compiled, Non-Standard Asset Names

The following listing shows what happens when `application.{js|css}` is renamed in the source to something else: Only `logo.png` is properly compiled.

**Precompile assets (after having renamed js and css files to something else—without fix). Execute in . /**

[Click here to view code image](#)

```
$ RAILS_ENV=production rake assets:precompile
I, [2015-02-22T20:45:47.062189 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/
logo-14e0571820ed8569b86519648f035ddd.png
```

The reason for this is that, by default, Rails' asset pipeline only compiles files named `application.{js|css}` (whether in subdirectories or not) into separate files in the compiled output. Every other filename is expected to be explicitly included somewhere within `application.{js|css}`.

Rails has a configuration option that allows non-standard filenames to be specified for separate compilation, namely `Rails.application.config.assets.precompile`.

If we wanted to name our engine asset files `app_component_engine.{js|css}`, we would add the following initializer to our `AppComponent` component:

**sportsball/components/app\_component/config/initializers/asset**

[Click here to view code image](#)

```
1 Rails.application.config.assets.precompile += [
2   'app_component/app_component_engine.js',
3   'app_component/app_component_engine.css'
4 ]
```

---

## Component Initializers

Components can have initializers just like Rails applications. Simply create the folder `config/initializers/` inside of your gem, and any Ruby code in there will be executed as the engine is initialized.

---

Now, after `rake assets:precompile`, we will get the desired output:

**Precompile assets (after having renamed js and css files to something else—with fix). Execute in . /**

[Click here to view code image](#)

```
$ RAILS_ENV=production rake assets:precompile
I, [2015-02-22T20:45:47.062189 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/
logo-14e0571820ed8569b86519648f035ddd.png
I, [2015-02-22T20:45:49.695917 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/
app_engine-6e039fbe90e5f75b1b3a88842d1face8.js
I, [2015-02-22T20:45:50.829624 #55331] INFO -- : Writing
```

```
./sportsball/public/assets/app_component/\  
app_engine-097b9ea4ccfc13ae1d38051c047dc0d.css
```

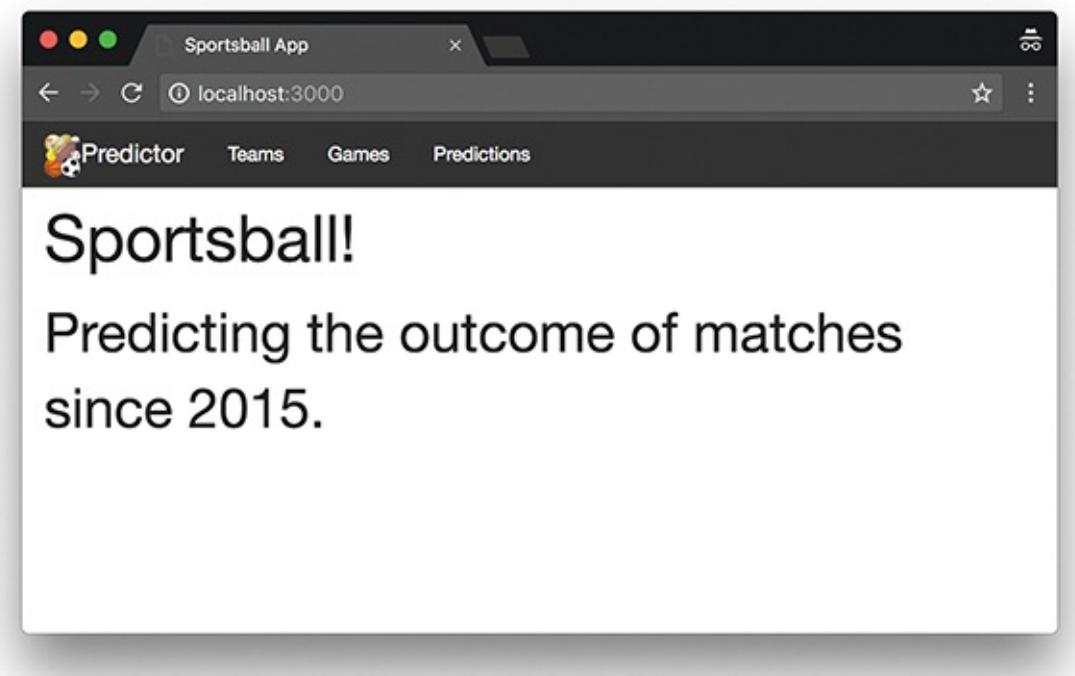
And the assets folder will contain the expected and correct content, namely, a compiled version of JS and CSS, and the application logo.

## Folder structure under ./public/assets/

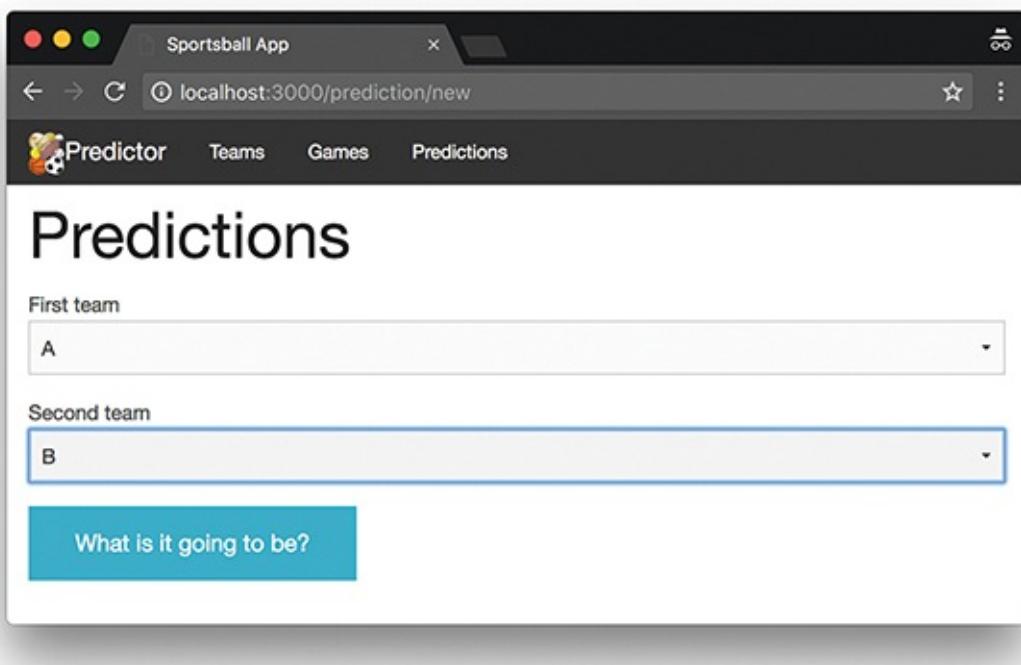
[Click here to view code image](#)

```
public/assets/  
└── app_component  
    ├── app_component_engine-097b9ea4ccfc13ae1d38051c047dc0d.css  
    ├── app_component_engine-097b9ea4ccfc13ae1d38051c047dc0d.css.gz  
    ├── app_component_engine-6e039fbe90e5f75b1b3a88842d1face8.js  
    ├── app_component_engine-6e039fbe90e5f75b1b3a88842d1face8.js.gz  
    └── logo-14e0571820ed8569b86519648f035ddd.png  
manifest-7a9f0159d72e2415368891385ba14b40.json
```

If we get everything right, the new look of Sportsball should be there whether we run the application in development or production. [Figures 3.5](#) and [3.6](#) show what it is supposed to look like.



**Figure 3.5.** Sportsball welcome page looking fly



**Figure 3.6.** Better-looking prediction page

## 3.4 Switching Databases

As it stands, Sportsball uses `sqlite` for storing data. `sqlite` is an in-process database that stores data to a file on disk. As such, we can run our app on some of our hardware or a virtual machine. However, we are not able to run it on a Platform-as-a-Service (PaaS) because of the transient nature of containers and their associated disk space. We would simply lose our application data at random points in time. If our app were running as multiple instances, the data between those instances would also start to differ with use. In effect, we must switch databases.

Rails makes switching databases straightforward in principle. This process remains simple as long as we are not dependent on non-standard database operations. Any SQL-compliant database would work for this example. Here we choose PostgreSQL.

Starting with the container application, we swap out PostgreSQL by changing both `Gemfile` and `database.yml`.

**./Gemfile with pg gem**

[Click here to view code image](#)

```
1 source "https://rubygems.org"
2
3 git_source(:github) do |repo_name|
4   repo_name =
5     "#{repo_name}/#{repo_name}" unless repo_name.include?("/")
6   "https://github.com/#{repo_name}.git"
7 end
8
9 path "components" do
10   gem "app_component"
11 end
12
```

```

13 gem "trueskill",
14   git: "https://github.com/benjaminleesmith/trueskill",
15   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
16
17 gem "rails", "~> 5.1.4"
18 gem "pg", "0.21.0"
19 gem "puma", "~> 3.7"
20 gem "sass-rails", "~> 5.0"
21 gem "uglifier", ">= 1.3.0"
22
23 gem "coffee-rails", "~> 4.2"
24 gem "turbolinks", "~> 5"
25 gem "jbuilder", "~> 2.5"
26
27 group :development, :test do
28   gem "rspec-rails"
29   gem "byebug", platforms: [:mri, :mingw, :x64_mingw]
30   gem "capybara", "~> 2.13"
31   gem "selenium-webdriver"
32 end
33
34 group :development do
35   gem "web-console", ">= 3.3.0"
36   gem "listen", ">= 3.0.5", "< 3.2"
37   gem "spring"
38   gem "spring-watcher-listen", "~> 2.0.0"
39 end
40
41 gem "tzinfo-data", platforms: [:mingw, :mswin, :x64_mingw, :jruby]

```

## **./config/database.yml**

[Click here to view code image](#)

```

1 development: &DEVELOPMENT
2   adapter: postgresql
3   database: sportsball_development
4   host: localhost
5   pool: 5
6   timeout: 5000
7
8 test: &TEST
9   <<: *DEVELOPMENT
10  database: sportsball_test
11  min_messages: warning
12
13 production:
14   adapter: postgresql
15   database: sportsball_production

```

Assuming that you have PostgreSQL running and that the current user has access rights to create databases, running `./test.sh` should already succeed.

Calling `rails db` will allow you to check that the database used by the application is now in fact PostgreSQL.

Assuming we are using PostgreSQL in production, we have now achieved dev/test/prod

parity for Sportsball (at least with respect to our database). However, AppComponent still uses sqlite for its tests, which means that for its test, we have not achieved this parity. This possibility of *partial* database dev/test/prod parity is a subtle way in which components complicate our application management.

Fortunately, the solution is simple and improves our ability to run database-dependent tests overall:

```
./components/app_component/app_component.gemspec
```

[Click here to view code image](#)

```
1 $:.push File.expand_path("../lib", __FILE__)
2
3 # Maintain your gems version:
4 require "app_component/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name      = "app_component"
9   s.version   = AppComponent::VERSION
10  s.authors   = ["Stephan Hagemann"]
11  s.email     = ["stephan.hagemann@gmail.com"]
12  s.homepage  = ""
13  s.summary   = "Summary of AppComponent."
14  s.description = "Description of AppComponent."
15  s.license    = "MIT"
16
17  s.files = Dir["{app,config,db,lib}/**/*",
18                "MIT-LICENSE", "Rakefile", "README.md"]
19
20  s.add_dependency "rails", "5.1.4"
21  s.add_dependency "slim-rails", "3.1.3"
22  s.add_dependency "jquery-rails", "4.3.1"
23  s.add_dependency "trueskill"
24
25  s.add_development_dependency "pg"
26  s.add_development_dependency "rspec-rails"
27  s.add_development_dependency "shoulda-matchers"
28  s.add_development_dependency "database_cleaner"
29  s.add_development_dependency "capybara"
30  s.add_development_dependency "rails-controller-testing"
31 end
```

```
./app_component/spec/dummy/config/database.yml
```

[Click here to view code image](#)

```
1 development: &DEVELOPMENT
2   adapter: postgresql
3   database: sportsball_app_component_development
4   host: localhost
5   pool: 5
6   timeout: 5000
7
8 test: &TEST
9   <<: *DEVELOPMENT
10  database: sportsball_app_component_test
```

```
11 min_messages: warning
```

Analogous to the container application, the component is changed to use the pg gem instead of sqlite and database.yml is adjusted appropriately.

Compare the database configuration of the main app and that of the component to realize that they are pointing at different database schemas: sportsball\_development and sportsball\_app\_component\_development. This change has two important implications. For one, tests now prove that a component truly only uses the database tables it (or one of its dependencies) defines, but no other tables potentially present in the app. This is not a common scenario—who would go around accessing database tables they didn't define if they don't have access to the models? Well, the bigger the codebase ... This is a property that grows more helpful with the size of the overall application. Furthermore, as the database connections of our component and app are now independent, they can be executed separately. They won't ever affect each other. Now we can, for example, run the tests of all components in parallel. Where previously the speed of test execution was a function of the overall size of the test suite, it becomes (within limitations) a function of the size of the slowest test suite of all components.

With this change, we are now ready to deploy to production!

## 3.5 Deploying to Platforms-as-a-Service

The intent is for the deployment of a CBRA application to be as simple as the deployment of a straight-up Rails application. I am going to discuss deployments to PaaS and not deployments to self-managed servers. In particular, I am going to cover Heroku (<https://www.heroku.com/>) as a leader in the public cloud space. I am also going to cover deploying to Cloud Foundry (CF), in particular, Pivotal Web Services (PWS) (<https://run.pivotal.io/>).

---

### Disclaimer

I currently work for Pivotal Software. Pivotal Software is a founding member of the Cloud Foundry Foundation (<http://www.cloudfoundry.org/index.html>), continues to be a major force behind the development of CF, and also operates PWS (<https://run.pivotal.io/>). I have personally participated in the development of one of CF's components, namely Loggregator (<https://github.com/cloudfoundry/loggregator>), its distributed logging infrastructure.

You get the gist: I am biased and I believe Cloud Foundry is a cool piece of software.

---

### 3.5.1 Deploying to Heroku

After setting up your account, install the Heroku toolbelt so you can log into Heroku on the command line to create and push applications. The full process is explained in Heroku's devcenter (<https://devcenter.heroku.com/articles/getting-started-with-ruby#set-up>).

The wonderful news from the deployment perspective is that CBRA applications behave exactly like any Rails application would. We can follow the default Rails deployment manual (<https://devcenter.heroku.com/articles/getting-started-with-rails4>). The manual goes over a couple of changes that a typical Rails will have to undergo to run on Heroku, and our application is no different:

- Fix the database: We have been using SQLite (<https://sqlite.org/>) as the database in development so far. Heroku requires that we not try to load the `sqlite3` gem when deploying to its services. We have two options:
  - Replace the `sqlite3` gem with `pg` gem and make PostgreSQL (<http://www.postgresql.org/>) our database in all environments. There is almost no exception to this being the right choice, as it is in support of “dev/prod parity” (<https://devcenter.heroku.com/articles/development-configuration#dev-prod-parity>), that is, keeping development and production environments as similar as possible.
  - The problem is also fixed by ensuring that `sqlite3` is only loaded in development, and `test`, but not in `production`. Inversely, load `pg` only in the `production` environment. If doing this, all tests should always be run against SQLite as well as PostgreSQL to ensure that both databases support all the features of the application.
- Make the app “12 factor”: Heroku subscribes to the principles of Twelve-Factor Apps (<http://12factor.net/>), which requires changes to Rails default behavior: the serving of static assets in production and the logging of messages to `stdout`. Adding the `rails_12factor` gem to the `Gemfile` takes care of both concerns.
- Set Ruby version: Next, not strictly required but advisable, is to specify the version of Ruby that is to be used for the app. Heroku wants us to add the following line to the top of `./Gemfile` to set Ruby to 2.4.2: `ruby '2.4.2'`

After pushing the application to Heroku with `git push heroku master`, we can migrate the database and our application is fully functional:

### **Deploy to heroku and run migrations. Execute in . /**

[Click here to view code image](#)

```
$ git push heroku master  
$ heroku run rake db:migrate
```

The current version of the application is deployed to Heroku at <https://shrouded-badlands-4039.herokuapp.com/>.

### **3.5.2 Deploying to Pivotal Web Services**

Deploying to PWS is fundamentally compatible to the deployment process to Heroku: PWS uses the same buildpack concept as Heroku and, in fact, it uses the same default buildpacks or forks of Heroku’s buildpacks.

In contrast to Heroku, PWS has a concept of organizations and spaces that provide a

hierarchy for applications and services to be deployed. CF as a technology is also inherently built to be deployed to many different infrastructures and locations. Because of these factors, the initial deployment of Sportsball is a bit more involved than when deploying to Heroku, and we will go over every command step by step.

## Deploy to PWS after the first login. Execute in . /

[Click here to view code image](#)

```
$ cf login -a https://api.run.pivotal.io
    #follow the instructions on screen
$ cf create-service elephantsql turtle sportsball_db
$ cf push sportsball --no-start
$ cf bind-service sportsball sportsball_db
$ cf start sportsball
```

Here is what is happening in these commands:

1. cf login kicks off a short interaction which has us enter our credentials. This authorizes our machine so we gain access to the account actions. -api targets a particular installation of CF. Since we are using PWS, we are targeting the API endpoint of its installation, which is <https://api.run.pivotal.io>.
2. cf create-service creates a service that can be bound to applications. There are a whole bunch of services available on the PWS marketplace (<https://console.run.pivotal.io/marketplace>). All we need in our case is to provision a PostgreSQL database.
3. cf push is the CF equivalent of git push heroku master. Unlike with Heroku, which acts as a git repository that an app is pushed to, CF uses this custom command. cf push takes all the contents of the current directory, zips them up, and sends them up to the service for deployment. Be aware that this means that the .gitignore file is ineffective at preventing content from being uploaded to CF. Create a .cfignore file for the same effect (<http://docs.cloudfoundry.org/devguide/deploy-apps/prepare-to-deploy.html>). --no-start ensures that the push does not attempt to start the application—that would not work as we still need to bind it to its database service.
4. cf bind-service links a service to an application and ensures that the application receives the necessary information about the service in environment variables so that the two can interact.
5. cf start triggers the starting of the application, after which it will be running, connected to all the services that have been bound.

### 3.5.2.1 Migrations

There is no equivalent to heroku run on CF and thus migrations must be run a different way. The CF Docs (<http://docs.cloudfoundry.org/devguide/services/migrate-db.html>) document three different ways to migrate a database, of which I am going to showcase the first: Running migrations from a local production environment. Buyers beware: CF gives you a powerful tool with this! Use it with caution!

A big difference between PostgreSQL as served by ElephantSQL (<http://www.elephantsql.com/>) and Heroku's PostgreSQL is that we can access it from anywhere once we have its location and login credentials. Exploiting this, we can point our local production database configuration and so allow us to simply run the migrations from our development machine as follows:

## Get environment variables with cf env. Execute in ./

[Click here to view code image](#)

```
$ cf env sportsball
Getting env variables for app sportsball in
    org sportsball / space development as shagemann@pivotal.io...
OK
```

System-Provided:

```
{
  "VCAP_SERVICES": {
    "elephantsql": [
      {
        "credentials": {
          "max_conns": "5",
          "uri": "postgres://rfghjuyt:qwertyuioplkjhgfdsa@babar
.elephantsql.com:5432/rfghjuyt"
        },
        "label": "elephantsql",
        "name": "sportsball",
        "plan": "turtle",
        "tags": [
          "Data Stores",
          "Cloud Databases",
          "Developer Tools",
          "Data Store",
          "postgresql",
          "relational",
          "New Product"
        ]
      }
    ]
  }
}
```

No user-defined env variables have been set

All the information we need is encoded into the service URI which has the key "uri." With this, we can change our production database configuration:

## ./config/database.yml - the production config section

[Click here to view code image](#)

```
1 production:
2   adapter: postgresql
3   database: rfghjuyt
4   username: rfghjuyt
5   password: qwertyuioplkjhgfdsa
6   host: babar.elephantsql.com
```

Again, this is a very potent tool; it is now more important than ever *not* to add `database.yml` to version control!

## Run migrations on production from the local machine. Execute in `./`

[Click here to view code image](#)

```
$ RAILS_ENV=production rake db:migrate
Migrating to CreateAppComponentTeams (20150110181102)
== 20150110181102 CreateAppComponentTeams: migrating =====
-- create_table(:app_teams)
 -> 0.0945s
== 20150110181102 CreateAppComponentTeams: migrated (0.0947s) =====

Migrating to CreateAppComponentGames (20150110184219)
== 20150110184219 CreateAppComponentGames: migrating =====
-- create_table(:app_games)
 -> 0.0722s
== 20150110184219 CreateAppComponentGames: migrated (0.0724s) =====
```

That's it! The current version of the application is deployed to CF at <https://sportsball.cfapps.io/>.

## 3.6 Updating Application Dependencies

Inevitably, once your app is in production and you are adding new features while making sure that everything stays healthy, there will be that moment when you will want to update some dependency. Now, in principle, updating dependencies works like it always has:

1. Update `dep`'s version in the `gemspec`.
2. Call `bundle update dep` to update to the new version

If you are following my advice to lock down dependency versions in your `gemspec` for external dependencies and if the gem is being used in several (or even many) of the app's components, this process is not quite as straightforward. Within *all* components, we will have to update the version and in *all* of them we have to update the bundle. If, on the other hand, you are not locking down gem versions in components, it is possible to update a gem only in the main app, while not updating the components. This causes your application to run different versions of your dependencies in prod than in component tests. If you want your customers to test your component versions for you, this is the way to go.

Again, I highly recommend locking down the versions of external dependencies in all components. If it has been the daunting task of updating all these versions at some point that has kept you from doing so, please read on to find a potential solution for this issue.

### 3.6.1 Updating the Bundle in All Components

First, let's tackle the problem of updating the bundles of all components, since this arises in any case. This task really boils down to executing `bundle update` `DEPENDENCY` for every component. In the CBRA group, a version of this with a nice API was presented by Enrico Teotti in the thread `gem updates in components` (<https://groups.google.com/d/msg/components-in-rails/iOQo2bdbdcQ/jB5H9Lmjtt8J>). This script exploits the assumption that all components will reside with a `/components` folder. If this is not true for you, this could easily be changed to point to a different directory.

#### Enrico's Bundle Update Enhancer (BUE)

[Click here to view code image](#)

```
1 #!/bin/bash
2
3 # BUNDLE UPDATE ENHANCER
4 # Preconditions:
5 # all components must have a Gemfile
6 # all components must use RVM and
7 #have a .ruby-version and .ruby-gemset
8
9 if [ $# -eq 0 ]; then
10    echo "No arguments supplied!"
11    echo "You must provide the name of the gem you want to update" +
12        "in your main app and all your components."
13    exit
14 fi
15
16 # ensure we bundle first
17 bundle
18
19 bundle show $1 > /dev/null
20 if [ $? -eq 0 ]; then
21
22    echo ">> BUE found $1 inside your main application. " +
23        "Updating it and its components now."
24    bundle
25    bundle update $1
26
27    echo ">> searching $1 inside the components"
28
29 # Iterate over the existing components
30 for component in components/* ; do
31
32    # Change directory to examined component
33    pushd $component > /dev/null
34
35    # Enable rvm
36    source "$HOME/.rvm/scripts/rvm"
37    rvm use $(cat .ruby-version)@$$(cat .ruby-gemset) --create \
38        > /dev/null
39
40    # ensure we bundle first
```

```

41     bundle
42     # bundle show is a good citizen and
43     # returns unix code 0 when successful.
44     bundle show $1 > /dev/null
45     if [ $? -eq 0 ]; then
46         echo ">> BUE found $1 inside $component. Updating it now."
47         bundle update $1
48     fi
49
50     # Move back directory.
51     popd > /dev/null
52 done
53
54 echo ">> All done"
55 else
56 echo "?? Nothing done, could not find $1"
57 fi

```

Alternatively, a different heuristic altogether is to find the locations of all files ending in .gemspec in the path of the app and to take the base directories of these. This changes the preceding script in lines 29 and 30 to the following:

[Click here to view code image](#)

```

1 for component_file in $(find . -name *.gemspec); do
2     component=`dirname $component_file`
```

### 3.6.2 Updating Every Component's Gem Version

One of the most common and strongest negative sentiments of people thinking about component-based Rails is: “If I have 20 components that all depend on Rails, then I will have 20 versions to update before I can update the version my app depends on.” To be frank, even in very large projects with many more components, I have not created an automated solution for this—although, that depends on your definition of “automated,” I guess.

The answer for this is twofold and lies in tooling and my convictions about dependency management. With respect to tooling, it is no secret that I like using the JetBrains (<https://www.jetbrains.com>) IDEs whenever I can (in fact, I am writing these lines with RubyMine right now) as I find them to far outperform any other editor in most situations, especially when I am working with a pair (in the sense of *programming partner*: one computer, two monitors, two keyboards, two mice) who might be new to a given language, framework, or maybe even programming in general. And since I lock all component versions, I can use its “Replace in Path” command (Cmd+Shift+R for me) to replace add\_dependency “dep”, “1.2.3” with add\_dependency “dep”, “1.2.5” for the entire app and all components with one command. This is as automated as I need it.

In addition to this automation, I find the problem of updating dependency versions too diverse a problem to try to come up with a mechanism that will “work in all situations.” I believe this mechanism doesn’t exist. How often does the update of one gem necessitate the update of another? How often is some part of the code dependent on a specific version

of a gem, a restriction that will take time to refactor out?

Another reason I don't think such a mechanism should exist is that dependencies create this pesky and tricky problem of my using code that I don't own, don't (fully) understand, and certainly don't know the future of ([http://www.theregister.co.uk/2016/03/23/npm\\_left\\_pad\\_chaos](http://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos)). Hence, I believe choosing to use a dependency and also updating it should be very conscious activities in any codebase.

### 3.6.3 Updating Rails

For the purpose of discussing how to upgrade Rails, there is a version of Sportsball using Rails 4.1.9 in the source code accompanying this book. This section will describe how the upgrade goes, if we update to Rails 5.0.0.1.

When we naively swap out these versions in the app's Gemfile and app\_component's gemspec, we get this result when calling bundle update rails:

**Naive and failing Rails update. Execute in . /**

[Click here to view code image](#)

```
$ bundle update rails
Fetching gem metadata from https://rubygems.org/.....
Fetching version metadata from https://rubygems.org/...
Fetching dependency metadata from https://rubygems.org/..
Resolving dependencies.....
Bundler could not find compatible versions for gem "actionmailer":
  In Gemfile:
    rails (= 5.0.0.1) was resolved to 5.0.0.1, which depends on
      actionmailer (= 5.0.0.1)

    app_component was resolved to 0.0.1, which depends on
      slim-rails (= 3.0.1) was resolved to 3.0.1, which depends on
        actionmailer (< 5.0, >= 3.1)
```

Now starts the non-mechanical process of figuring out—and fixing—incompatible gem versions. In our case, slim-rails 3.0.1 is incompatible with Rails 5. Updating it to the latest version means adding version 3.1.0 to the gemspec of app\_component. Running the same command again, we find the culprit to be jquery-rails. After updating it, we find a problem with Gemfile.lock and realize that we need to resort to the less targeted bundle update, which will update any gem that is updateable within the restrictions of the dependency graph. We continue our fixes with sass-rails (5.0.6), coffee-rails (4.2.1) before bundle update runs through successfully and we have not only updated these gems but a whole slew of others (most dependencies of the app, in fact).

---

## Healthy Gemfiles

While we have not previously focused on the gem versions in the application's Gemfile, this issue comes up now. Jon Barker has a nice essay on healthy

gemfiles (<https://blog.pivotal.io/labs/labs/the-healthy-gemfile>) in which he makes the following observations:

- Locking down gem versions quickly makes an app stale.
- Dependency versions are locked down via Gemfile.lock (no need to do so doubly, by locking them in the Gemfile as well).
- If not general practice, a locked-down version in the Gemfile sends a signal. Add a comment as to why it is locked down (e.g., “we depend on this version because of this bug [include link]. Once fixed, we should be able to update”).

I am in total agreement with Jon on gemfiles in applications; however, for all the reasons stated before, I believe that we need to lock down versions of our direct dependencies in our components.

---

We shouldn't expect tests to work after this update—we jumped a Rails major version, after all. For the following, I am also going to ignore a whole host of deprecation warnings and focus on getting the tests green again. In running the tests for the component, we get the info that controller test helpers have been extracted into the gem rails-controller-testing. For it to work, we require 'rails-controller-testing' in spec\_helper.rb and add the following RSpec config:

**./components/app\_component/spec/spec\_helper.rb (partial) to fix controller tests**

[Click here to view code image](#)

```
1 RSpec.configure do |config|
2
3   ...
4
5   [:controller, :view, :request].each do |type|
6     config.include ::Rails::Controller::Testing::TestProcess,
7       :type => type
8     config.include ::Rails::Controller::Testing::TemplateAssertions,
9       :type => type
10    config.include ::Rails::Controller::Testing::Integration,
11      :type => type
12  end
13 end
```

This leaves us with one failure in app\_component when running test.sh:

**./test.sh still failing. Execute in ./components/app\_component**

[Click here to view code image](#)

- 1) the prediction process get a new prediction  
Failure/Error: visit '/app\_component/'  
ActionView::Template::Error:  
Asset was not declared to be precompiled in production.  
Add `Rails.application.config.assets.precompile +=  
 %w( app\_component/logo.png )` to  
`config/initializers/assets.rb` and restart your server

Since we are working on an engine, we don't want to follow this instruction verbatim, but rather add the requested precompile path to our engine's initializers. One way to do this is to update `engine.rb` to add an initializer block.

```
./components/app_component/lib/app_component/engine.rb
```

[Click here to view code image](#)

```
1 module AppComponent
2   class Engine < ::Rails::Engine
3     isolate_namespace AppComponent
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11
12    initializer :assets do |app|
13      app.config.assets.precompile += %w( app_component/logo.png )
14    end
15
16    config.generators do |g|
17      g.orm             :active_record
18      g.template_engine :slim
19      g.test_framework  :rspec
20    end
21  end
22 end
```

Running `app_component`'s tests again, we discover another failure, which can be traced to a problem between `capybara` and `rack` (<https://github.com/jnicklas/capybara/issues/1592>), which has been fixed in the versions of `capybara` greater than 2.7.1. This fixes all tests for the component and, as you will find, also fixes all tests for the entire application.

I spelled out the individual steps and surprises of updating Rails to illustrate how many issues in this process are *not* related to the actual updating of the dependency's version number, but rather to the necessitated changes in code. I hope this convinces the last doubters that managing dependency versions works just as fine in component-based Rails applications as it does in “standard” ones.

### 3.6.4 Long-Running Dependency Updates

In situations where the update process for gem seems an insurmountable effort, like updating a major version of Rails seems to be for many apps, there is another helpful practice. The practice unhinges the recommendation of locking down dependency versions to create a smoother update path.

In the `gemspec`, we add the following condition for a difficult-to-update dependency.

```
./components/app_component/app_component.gemspec (partial)
```

[Click here to view code image](#)

```
1 if ENV['RAILS_5_UPDATE']
2   s.add_dependency "rails", "5.0.0.1"
3 else
4   s.add_dependency "rails", "4.1.9"
5 end
```

As we go down the path of updating Rails in this case, we are going to add more conditional versions—just like we saw in updates to additional gems while updating Rails. With this in place, we can now create a new test runner, say `test_rails5.sh`:

`./components/app_component/test_rails5.sh`

[Click here to view code image](#)

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running app component engine specs FOR RAILS 5"
6 RAILS_5_UPDATE=true bundle exec rake db:create db:migrate
7 RAILS_5_UPDATE=true RAILS_ENV=test bundle exec \
8   rake db:create db:migrate
9 RAILS_5_UPDATE=true bundle exec rspec spec
10 exit_code+=$?
11
12 exit $exit_code
```

With these mechanics in place, we can add `test_rails5.sh` as a separate build to our build box and slowly introduce updates that will create Rails 5 compatibility. We are in full control of how fast or slow we want to do the migration and can keep the existing version working for as long as we need to.

### 3.6.5 Listing a Dependency Only Once

If I have not been able to convince you that managing the actual version numbers is your smallest problem, here is one last pattern you could use. Again, I have never done anything else other than manage these versions for all components “by hand” as described before.

It turns out that all you need to create a gem is a `gemspec`. Take the following directory structure and content for `rails_version.gemspec`.

#### Hypothetical gem `rails_version`

[Click here to view code image](#)

```
./sportsball
└── components
    └── app_component
        └── ...
    └── rails_version
        └── rails_version.gemspec
```

`./components/rails_version/rails_version.gemspec`

[Click here to view code image](#)

```
1 $:.push File.expand_path("../lib", __FILE__)
```

```

2
3 # Describe your gem and declare its dependencies:
4 Gem::Specification.new do |s|
5   s.name          = "rails_version"
6   s.version       = "0.0.1"
7   s.authors       = ["The CBRA Book"]
8   s.summary        = "CBRA component"
9
10  s.files = []
11  s.test_files = []
12
13  s.add_dependency "rails", "5.1.4"
14 end

```

bundle will *not* complain if we require this gem from within app\_component and the container application. This means that instead of requiring Rails directly from within every component, we can just require this *gem* of a gem and have only one place in which the version of Rails is directly mentioned.

Extrapolating from this, we might be tempted to put all versions of all gems we use anywhere in the app into this gem so there is only one place where we need to mention *any* version of *any* gem. While possible, it also removes the benefit from one of the biggest drivers behind introducing components in the first place—to reduce unnecessary dependencies where possible.

## 3.7 Proposal for a Different Application Root—Showcasing the Difference of Components’ Structure

What we have done so far is quite a radical departure from the classic way developers think about and like to write Rails applications. We have reduced the actual Rails application to a shell that only glues a component into the right place to execute it. That component does all the work, embodies the domain, and frankly, between the two, is the much more interesting part.

Yet, when we look at the root of the application, it still very much looks like standard Rails. Sure, the app folder is gone and there is a conspicuous new folder called components, but almost everything else is still standard Rails.

### The root folder of Sportsball

[Click here to view code image](#)

```

sportsball
├── bin
├── components
├── config
├── db
├── lib
├── log
├── public
├── spec
├── tmp
└── vendor

```

```
└── Gemfile
└── Gemfile.lock
└── README.md
└── Rakefile
└── build.sh
└── config.ru
└── package.json
└── test.sh
└── vendor.tree
```

What if we wanted to change that?

### 3.7.1 Why Would I Want to Change That?

In 2011, Robert Martin gave a keynote at Ruby Midwest titled “Architecture the Lost Years” (<https://www.youtube.com/watch?v=WpkDN78P884>). Put this book down now and watch his talk if you have not already done so.

Welcome back! Before we continue, let us briefly recap the main takeaways from the keynote:

1. Standard Rails makes the structure of an auxiliary function, namely *Web delivery*, very obvious, while it hides away the structure of the domain that is being delivered. To paraphrase him: Standard Rails applications don’t scream their *intent* at you, they scream their *framework* at you.<sup>5</sup>
2. Rails’ “convention over configuration” approach has the unintended side effect of dissuading developers who know nothing else from looking for useful patterns and structures beyond those offered by Rails.
3. The delivery mechanism should be a “plugin” to the application.
4. The database is a detail that might be deferred by good architecture. And more generally: Standard Rails’ powerful set of defaults pushes developers in the direction of opting for things they might not need.

One observation underlies all these points: Everything that is very successful or popular must, to a certain extent, crowd out alternatives that are less successful and popular. All these statements about Rails are statements about “what also happened” because Rails turned out to be so very useful.

In the remainder of this book, we will discuss how Robert Martin’s points can be mitigated through the use of component-based Rails architectures. This section starts us off by attacking his first point: Changing what is perceived as important within the application structure.

### 3.7.2 How CBRA Can Change How I Present and Perceive My App

How does one go about tackling Robert Martin’s first point? How do we start to reduce the perceived importance of the Web delivery framework?

The main app interacts with the `AppComponent` component in only two places: `Gemfile` and `routes.rb`. Any change to their relationship should therefore be

straightforward. In fact, when it comes to filesystem locations, only the `path` option in the `Gemfile` is affected. So, we should be able to move the main app so that the root of our project does not contain all this Rails stuff!

## Move the container app into a subfolder. Execute in `./`

[Click here to view code image](#)

```
$ mkdir web_container  
$ mv * web_container  
$ mv web_container/components .
```

And indeed, if we move all the visible files and folders in the application root directory except for `components` into a new folder `web_container`, we get something that looks like this:

## Structure of Sportsball with the main app pushed down into `web_container`

[Click here to view code image](#)

```
sportsball  
└── components  
    └── app_component  
└── web_container  
    ├── bin  
    ├── config  
    ├── db  
    ├── lib  
    ├── log  
    ├── public  
    ├── spec  
    ├── tmp  
    ├── vendor  
    ├── Gemfile  
    ├── Gemfile.lock  
    ├── README.md  
    ├── Rakefile  
    ├── build.sh  
    ├── config.ru  
    ├── package.json  
    ├── test.sh  
    └── vendor.tree
```

To run any of the Rails commands, we need to first `cd` into the `web_container` directory. Right now, the app is broken; if we try to run `rails s`, we get the following error:

## Running Rails is broken. Execute in `./web_container`

[Click here to view code image](#)

```
$ rails s  
The path `./web_container/components/app_component` does not exist.
```

The only change necessary to fix this issue is to add “`.. /`” to the beginning of the relative path by which the `AppComponent` component is referenced from the container app’s `Gemfile`.

```
./web_container/Gemfile new path block directory
1 path ".../components" do
```

Voilà! The app works again. And the root directory no longer looks like a standard Rails app. The first point from Robert Martin's list is taken care of! Well, at least to the extent that we have removed the “screaming of the Web framework” part. What is missing is to make the app “scream” its true intent in a better way. The path toward that is straightforward, however: Splitting the AppComponent component into multiple components that make sense will slowly make more structure visible.

---

## .gitignore After Change

When performing this check, `.gitignore` will need to be double- checked. At least Rails' default file will no longer ignore SQLite and log files.

---

While I personally like this change a lot, it is not yet common in the wild. That is certainly due in part to the expectations that CI services and PaaS have about Rails apps, which make this structure not work out of the box. So, let's fix Travis CI, Heroku, and CF to work with the new structure.

### 3.7.3 Making CI and PaaS Work with the New Structure

In this section, we will fix the continuous integration solutions we discussed in [Section 3.2](#) and the PaaS we discussed [Section 3.5](#).

#### 3.7.3.1 Continuous Integration: Travis CI

The needed change to `.travis.yml` is minor: All there is to do is to fix the location of the `Gemfile` of the container app, like so:

```
./.travis.yml
```

[Click here to view code image](#)

```
1 gemfile:
2   - web_container/Gemfile
3   - components/app_component/Gemfile
4 script:
5   - ./test.sh
6 before_install:
7   - cd $(dirname $BUNDLE_GEMFILE)
8 rvm:
9   - 2.4.2
```

When run, however, the specs of the main app fail with a strange error for Sportsball.

#### Strange Travis CI error when moving the container app

[Click here to view code image](#)

Failures:

- 1) Engine 'AppComponent' hooks up to /  
Failure/Error: visit "/"  
NoMethodError:

```

undefined method `visit' for
#<RSpec::ExampleGroups::EngineApp:0x00000002524390>
# ./spec/features/app_spec.rb:3:in
# `block (2 levels) in <top (required)>`
Finished in 0.00062 seconds (files took 0.21662 seconds to load)
1 example, 1 failure
Failed examples:
rspec ./spec/features/app_spec.rb:2
  # Engine 'AppComponent' hooks up to /
The command "./test.sh" exited with 1.
Done. Your build exited with 1.

```

The error hints at capybara, which defines visit, not being loaded. In fact, no spec setup has taken place, because our RSpec setup relied on our use of the .rspec file, which needs to be in the root of the project. Since we only moved the visible files, .rspec is still in the root of the repository, but no longer in the root of the container app. Once moved down one level, .rspec is once again in the correct location and the specs will pass as before.

### 3.7.3.2 PaaS Deployment: Heroku and CF

Just like in [Section 3.5](#), Heroku and Cloud Foundry are going to be our examples of how to deploy to PaaS. It turns out that the different approaches these two platforms take to pushing code make for very different approaches to pushing our Rails application with a non-standard folder structure.

CF and Heroku use very similar buildpacks for Ruby—CF's buildpack (<https://github.com/cloudfoundry/ruby-buildpack>) is a fork of Heroku's (<https://github.com/heroku/heroku-buildpack-ruby>). Both buildpacks expect that Ruby and Rails applications be in root: They both detect applications by checking for the presence of Gemfile in the root of the pushed directory structure.

If we just try to push Sportsball with its new structure from the root directory, Heroku and CF are going to be unable to detect our application:

#### **Deploy to Cloud Foundry failing to detect the Sportsball application. Execute in . /**

[Click here to view code image](#)

```

$ cf push sportsball
Updating app sportsball in org sportsball / space development
  as stephan...
OK

```

```

Uploading sportsball...
Uploading app files from: ./sportsball
Uploading 4.6M, 922 files
OK

```

```

Stopping app sportsball in org sportsball / space development
  as stephan...
OK

```

```
Starting app sportsball in org sportsball / space development
  as stephan...
OK
----> Downloaded app package (4.9M)
```

FAILED

```
Server error, status code: 400, error code: 170003, message:
  An app was not successfully detected by any available buildpack
```

The good news is that both Heroku and Cloud Foundry support pushing our new application structure with only a little extra work. Basically, we are going to create a copy of the application for deployment. We are going to give this copy the standard Rails application folder structure. Finally, we are going to upload this deploy artifact to the PaaS with the appropriate mechanism.

The deploy directory we create is the same for both PaaS. To create it, we perform the following steps:

1. Create a deploy directory and copy the to-be-deployed Rails application into it.
2. Copy the components directory into the deploy directory so that it becomes a subdirectory.
3. Change the Gemfile and the Gemfile.lock to expect the components not at .. /components, but instead at ./components.

We have wrapped these steps into a shell script `prepare_deploy_directory.sh`. Note that we are using sed (<https://www.gnu.org/software/sed/manual/sed.html>) to change the references to the component directory in both Gemfile and Gemfile.lock (see lines 12 through 14 in the following).

**`./prepare_deploy_directory.sh`**

[Click here to view code image](#)

```
1 #!/bin/bash
2 trap "exit" ERR
3
4 echo "      Copy deploy files into place"
5 rm -rf deploy
6 mkdir deploy
7 cp -R web_container/ deploy
8 cp -R components deploy
9 rm -rf deploy/tmp/*
10
11 echo "      Fix components directory reference"
12 sed -i -- s|\.\./components|components|g deploy/Gemfile
13 sed -i -- s|remote: \.\./components|remote: \.\./components|g \
14   deploy/Gemfile.lock
15
16 echo "      Uploading application..."
```

With the deploy directory in place, pushing our latest code to CF is very straightforward. Let's do this via another script, `deploy_to_cloudfoundry.sh`. It

will be nice to have this around, as we will always have to recreate the deploy directory whenever we want to push and we can make sure that happens within the script (line 7). In lines 9 and 10, the script simply navigates into the deploy directory and calls cf push to invoke the CF command line interface to push the deploy folder.

**./deploy\_to\_cloudfoundry.sh**

[Click here to view code image](#)

```
1 #!/bin/bash
2 trap "exit" ERR
3
4 APP_NAME="sportsball"
5
6 echo "-----> Deploying to Cloud Foundry"
7 prepare_deploy_directory.sh
8
9 cd deploy
10 cf push $APP_NAME
11 cd ..
```

---

## CF Deployment Via bundle package, Plus This One Weird Trick

Courtesy of Arjun Sharma comes a second strategy for deploying applications with this structure. bundle package --all vendors all gems, including gems referenced via path.

If this is all we do, cf push errors with the following message:

[Click here to view code image](#)

```
Running: bundle install --without development:test
--path vendor/bundle --binstubs vendor/bundle/bin -j4
--deployment
  You are trying to install in deployment mode after changing
  your Gemfile. Run `bundle install` elsewhere and add the
  updated Gemfile.lock to version control.
  You have added to the Gemfile:
  * source: source at ../components/app_component
  You have deleted from the Gemfile:
  * source: source at vendor/cache/app_component
```

The issue is that once we run bundle package, the Gemfile.lock changes to say:

[Click here to view code image](#)

```
1 PATH
2   remote: vendor/cache/app_component
3   specs:
4     app_component (0.1.0)
```

But the Gemfile still says:

[Click here to view code image](#)

```
1 gem 'app_component', path: '../components/app_component'
```

Because of this, bundle install —deployment will complain about the mismatching paths. This turns out to be known issue #2175 with bundler (<https://github.com/bundler/bundler/issues/2175>). The issue thread gives us a workaround, in which we change the Gemfile so that it sets the path to the local path if that path is available and points it at vendor/cache otherwise.

[Click here to view code image](#)

```
1 gem 'app_component',
2   path: (Pathname.new('../components')).exist? ? '../component\'
3 s/app_component' :
4   'vendor/cache/app_component')
```

---

There are situations in which git submodule or git subtree can be used to get around this problem; this can effectively reduce the repository to the application subdirectory and upload this to the PaaS.<sup>6</sup> The major problem we have created, and the reason that it does not work for us, is that web\_container is no longer “self-contained”—it no longer contains all of the code it needs (namely, components, which lives outside of it).

Things are not quite as easy for a deploy to Heroku, as it uses git as a way of pushing applications. To continue to use this mechanism, we would have to somehow create a git repository only for the deploy directory, check in the code, and then push from within that directory. While that is possible, I want to discuss a different approach here—one that does not involve messing with git.

There is an alternative to git for pushing applications accessible through the Heroku Platform API (<https://devcenter.heroku.com/articles/platform-api-reference>). The Platform API is a collection of JSON endpoints for the management of almost everything that is client facing at Heroku. The particular part of this API that we are interested in is the Build API (<https://devcenter.heroku.com/articles/build-and-release-using-the-api>), which deals with the building and releasing of applications.

To deploy an application, we call the builds endpoint at [https://api.heroku.com/apps/APP\\_NAME/builds](https://api.heroku.com/apps/APP_NAME/builds) with a POST. This endpoint expects the URL of a tarball archive of the application we are deploying. While we could put our application anywhere on the public web and send its URL along with this call, Heroku also provides a sources endpoint to which we can upload our tarball in a secure way to be used for the deploy.

The following script performs this list of activities for us:

[./deploy\\_to\\_heroku.sh](#)

[Click here to view code image](#)

```
1#!/bin/bash
2trap "exit" ERR
3
4APP_NAME="stormy-hollows-9630"
5
```

```

6 echo "-----> Deploying to Heroku"
7 prepare_deploy_directory.sh
8
9 VERSION=`git rev-parse HEAD | perl -pe "chomp"`
10 echo "-----> Deploying application version $VERSION"
11
12 echo "          Creating build tarball...."
13 DEPLOY_FILENAME="deploy-$VERSION.tgz"
14 pushd deploy
15 tar -czf ../$DEPLOY_FILENAME .
16 popd
17
18 echo "          Requesting application specific source endpoint..."
19 acceptheader="Accept: application/vnd.heroku+json; "
20 acceptheader+="version=3.streaming-build-output)"
21 SOURCE_ENDPOINT=$(curl -s -n \
22     -X POST "https://api.heroku.com/apps/$APP_NAME/sources" \
23     -H $acceptheader)"
24
25 PUT_URL=`echo $SOURCE_ENDPOINT | jsawk \
26     "return this.source_blob.put_url"`
27 echo "          Received blob endpoint: $PUT_URL"
28 GET_URL=`echo $SOURCE_ENDPOINT | jsawk \
29     "return this.source_blob.get_url"`
30 echo "          Received deploy endpoint: $GET_URL"
31
32 echo "          Upload app blob"
33 curl -s "$PUT_URL" -X PUT -H "Content-Type:" \
34     --data-binary @$DEPLOY_FILENAME
35
36 echo "          Deploy application"
37
38 data={"source_blob": {"url": "$GET_URL"}, " "
39 data+="version": "$VERSION"} }
40 DEPLOY_RESULT=$(curl -n
41     -X POST "https://api.heroku.com/apps/$APP_NAME/builds" \
42     -d $data
43     -H $acceptheader -H "Content-Type: application/json") "
44
45 log_url=`echo "$DEPLOY_RESULT" | \
46     jsawk "return this.output_stream_url"`
47 echo "          Received log endpoint: $log_url"
48
49 curl "$log_url"

```

Here is what the script is doing, step by step:

- Line 15 creates the tarball that holds the deploy artifact from the `deploy` directory we created earlier.
- Lines 19 through 23 invoke the `sources` to get a new, temporary, secure “source URL” for the upload of our build tarball.
- Lines 23 through 29 are using `jsawk` (<https://github.com/micha/jsawk>) to extract the source URL from the JSON response of the previous call (we get both an upload and a download version of this URL).

- Lines 33 and 34 uploads the tarball to the source URL. After this call, the download version of the source URL will return our application tarball.
- Finally, lines 38–43 uses the source URL to deploy the application using the builds endpoint. In its response, the call returns a streaming log URL, which we extract in lines 45 and 46 so that we can use it in line 49.
- In line 49 we use `curl` to see the progress of the deploy as it is happening.

In the end, deploying like this may not feel as nice as deploying via `git push heroku master`, but it gives us back the ability to decide our application’s structure and make it the best we can for the task at hand instead of being forced into a one-size-fits-all solution.

### 3.7.4 What Is Next?

A peculiar thing would happen if everyone started using this pattern of having two folders, `components` and `web_container`, in the root of projects: It would be very simple to detect CBRA applications, as their loudest message is now the particular dualism in their structures. In comparison to the original situation, however, the situation is much more flexible: One could use different names for both those folders if different names work better in a particular context. One could split up the components into multiple subfolders if one detects patterns in the types of components. Instead of one `web_container`, one could have multiple Rails apps in the root folder, maybe under `web_containers` or directly in the root folder.

The remaining gigantic caveat to everything we have stated is that we currently only have the `AppComponent` component in the `components` folder! We can hardly say that we are giving high-level information regarding whatever it is that the app is about. How to find and create more components is the topic of the next chapter.

1. The routes issue has been discussed and solved by Starr Horne (<http://blog.honeybadger.io/rails4-engine-controller-specs/>), Ben Smith (<https://content.pivotal.io/blog/writing-rails-engine-rspec-controller-tests>), and Ryan Bigg (<https://stackoverflow.com/questions/5200654/how-do-i-write-a-rails-3-1-engine-controller-test-in-rspec>).
2. If your engine is called `CrazyEngineName`, its snake-case version is `crazy_engine_name`.
3. In the second version of the `PredictionHelper` spec, the `describe` block is nested inside of the block created by `module AppComponent`, which means that the `describe` (and everything therein) is within the lexical scope of the `AppComponent` module.
4. To find the logs, navigate to the project page for Sportsball and then click on the red dot next to the build you are interested in. This brings you to the console page, which contains everything that the build process sent to `stdout`.
5. Check at around 10:30 in the video linked above. I do disagree with some of the details of the further discussion. At around 28:00, Robert Martin goes on to ask, “What would the names of those directories be? Interactors, entities, and boundaries. Excellent. If you go into the interactors, what would you see? `create_order`. So, at the very top of the application you would see a structure that would start to look like the intent of the application.” With this, the top-level structure once again does not contain the defining names, the intent of the application. Once again, it contains technical constructs. From my perspective, it is a good thing that nothing stands in the way of flattening out the directory structure and elevating the concept names to the top of the structure.
6. Using `git subtree` (<https://github.com/apenwarr/git-subtree/blob/master/gitsubtree.txt>), repositories that hold multiple self-contained applications in subdirectories can be pushed to Heroku via `git subtree push --prefix APP_FOLDER heroku master` and to CF by calling `cf push` from within the subdirectory.

## CHAPTER 4

# Component Refactorings: Extracting Components out of Components



Photo: Eric Isselee/Shutterstock

With the previous chapter behind us, we know how to create a component-based application. It covered all the techniques relevant in the creation of a component. If, however, you picked up this book because you would like to change the existing application you are working on, there is a big, unanswered question: How do you *find* the actual components in your application?

Also, with just one component present, the benefits of CBRA are impossible to communicate. We have already discovered many quirks when using gems and engines as components, and for what benefit? So far the only achievement is the separation of the Rails container from the code directly belonging to the application. Would we really want to call that, a system consisting of one component only, a *component-based system*? Probably not. The old thing was replaced by a new thing, but it is still just *one thing*. A component wants siblings. A component wants more components with which it can interact—components that, together, can be assembled into a complex system.

This chapter will start to investigate this question by showing the extraction of multiple components out of what is currently a one-component app. And whether you are

extracting components out of other components or out of applications, the process is the same.

## 4.1 Determining What to Extract: Bottom-Up

Finding the first thing to extract is not easy. We are so used to everything being “together” that merely asking the question may even seem absurd to some. What we need to be looking for are the seams of the application: What are the parts of the application where the connections are easiest to reverse?

In a way, a non-componentized codebase is like a cloth made from felt. Individual fibers are visible, but nothing seems to be separable. Yet if you start pulling on it (and the felt isn’t too thick), it will start coming apart along some line that was previously hidden.

If we try to apply a similar idea to our codebase, the question becomes: Can we see indications of seams in code? Or can we derive the likely position of seams from the application domain?

I believe that these two options are both possible and complementary to each other. Looking at the code and reasoning about it is a bottom-up approach, while reasoning about the domain and trying to understand where the boundaries might lie is a top-down approach.

Let’s try the bottom-up approach first and see what surfaces. Let’s also focus on Ruby for now—while in almost every app we will also find JavaScript, HTML or some variant, SQL, and bash, it is always true that we are looking at a Ruby application. As an exercise, the reader may put the book aside and open the source code to Sportsball (<https://github.com/shageman/component-based-rails-applications-book/tree/master/c3s03>) as it stood in [Section 3.3](#). Start skimming through the source code inside of `components/app_component/app`, navigating the entire tree from top to bottom:

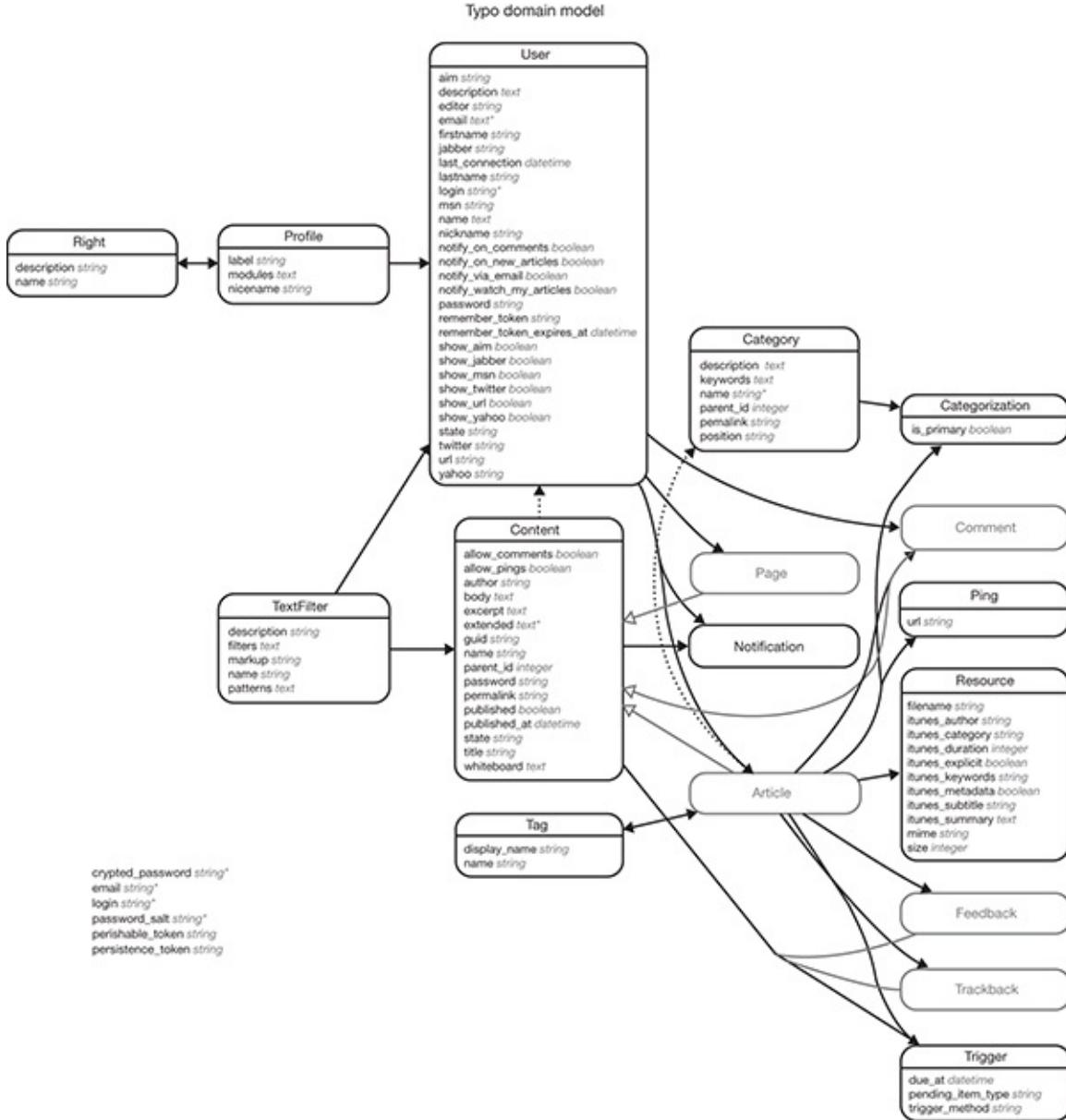
---

### Using Entity-Relationship Models to Find Seams

Using the relations between classes stored in the database is another interesting way to look for seams within the applications. Unfortunately, as previously discussed, some of the “standard” Rails literature creates messy object relationships because new relationships are not being scrutinized (as something that might come back to haunt us later), but rather are opportunistically introduced as something that is “just good to have a lot of.” Unfortunately, there is no value in pursuing this investigation for Sportsball, as there are only two entities. So, this will have to be a theoretical endeavor.

Whether the object diagram leads to anything interesting is easily checked by drawing it—its entity relationship diagram (ERD). The Rails ERD gem (<https://github.com/voormedia/rails-erd>) can do this for us. Rails ERD’s own gallery (<https://voormedia.github.io/rails-erd/gallery.html>) contains a couple of examples that are worthwhile to study.

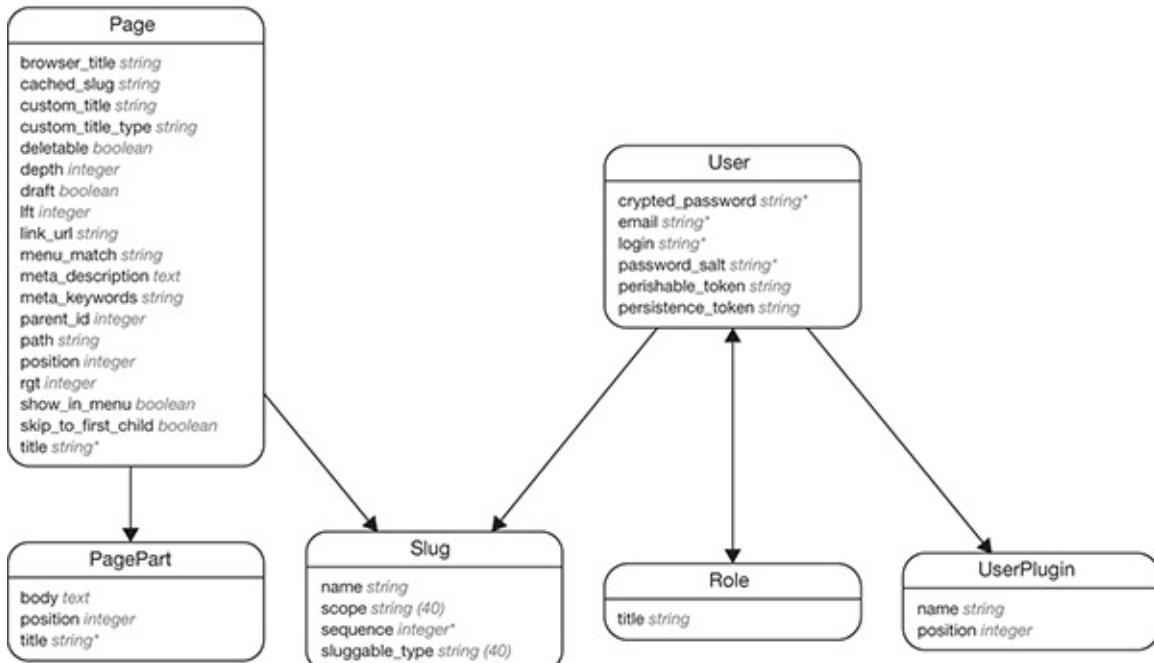
Consider, for example, the ERD of an older version of Publify (formerly Typo) (<https://github.com/publify/publify/wiki>) depicted in [Figure 4.1](#). It is hard to make out any parts that are not, or just *less*, connected. From this diagram, Textfilter merits a first look, as it references other objects, but others don't reference it. Similarly, Trigger is only being referenced by others, but has no references itself.



**Figure 4.1.** Rails ERD output for Publify (formerly Typo)

RefineryCMS's (<http://www.refinerycms.com/>) ERD is shown in [Figure 4.2](#). While again no disconnected parts are visible, the two large boxes, page and user, are of note. This being a CMS, both page and user can be expected to be important concepts in the application. Interestingly, page only references others but is not itself referenced. This is not true for user, which has a circular reference with role (in my experience, a common occurrence for these two concepts). I recommend investigating whether there might be three components here: Pages (with Page and PagePart), Slugs (with Slug), and Users (with User, Role, and UserPlugin).

## Refinery domain model



**Figure 4.2.** Rails ERD output for RefineryCMS

---

### Current directory structure of ./component/app\_component/app

[Click here to view code image](#)

```

./sportsball/components/app_component/app
├── assets
│   ├── config
│   │   └── app_component_manifest.js
│   ├── images
│   │   └── app_component
│   │       └── logo.png
│   └── javascripts
│       └── app_component
│           └── application.js
└── stylesheets
    └── app_component
        └── application.css

```

```

controllers
└── app_component
    ├── application_controller.rb
    ├── games_controller.rb
    ├── predictions_controller.rb
    ├── teams_controller.rb
    └── welcome_controller.rb

```

```

helpers
└── app_component
    └── predictions_helper.rb

```

```

models
└── app_component
    ├── application_record.rb
    ├── game.rb
    ├── prediction.rb
    ├── predictor.rb
    └── team.rb

```

```

└─ views
    └─ app_component
        └─ games
            └─ _form.html.slim
            └─ edit.html.slim
            └─ index.html.slim
            └─ new.html.slim
            └─ show.html.slim
        └─ predictions
            └─ create.html.slim
            └─ new.html.slim
        └─ teams
            └─ _form.html.slim
            └─ edit.html.slim
            └─ index.html.slim
            └─ new.html.slim
        └─ welcome
            └─ show.html.slim
    └─ layouts
        └─ app_component
            └─ application.html.slim

```

`assets` does not contain any Ruby, so we can skip it for now. Skim through `controllers` and notice that every file here depends on `ApplicationController`, which in turn depends on `ActionController::Base`—there are no clear seams here—these relationships are true for all controllers in this folder. `helpers` contains two files, one of which is without content, while the other contains one 1-line method:

**`./components/app_component/app/helpers/app_component/predictionsHelper.rb`**

[Click here to view code image](#)

```

1 module AppComponent
2   module PredictionsHelper
3     def prediction_text(team1, team2, winner)
4       "In the game between #{team1.name} and #{team2.name} " +
5       "the winner will be #{winner.name}"
6     end
7   end
8 end

```

What is interesting for our analysis is that this file is self-contained. It mentions no constants that are not defined within the file itself. Its only expectation about the world is that any caller calls `prediction_text` with three parameters, which need to be constructs that name can be called on.

Ideally, when extracting code, we find it in this state of self-containedness. More often than not, however, we have to perform some refactorings to remove a few instances of knowledge of other parts of the system. We will discuss this more later, and for now the question is: Did we find our first extractable component?

This file is probably not our first component in that it does not encapsulate a reasonably interesting and sizable part of our domain. Really, the right refactoring for this method is to roll it into the view layer (it was extracted as a helper method to have an example of

testing helpers in [Section 3.1](#)).

While we have not found our first component, we now know what we are on the lookout for: (almost fully) self-contained files that indicate that they are dealing with separate parts of the domain.

The next folder to tackle is `models`. The first file therein, `Game`, is not self-contained: It depends on `ActiveRecord::Base` and mentions `Team`. The next file is `Prediction`, which is self-contained—no mention of anything not defined in the file, but once again not really a sizable part of our domain. `Predictor` is not self-contained: It depends on `Saulabs::TrueSkill::Rating`, `Saulabs::TrueSkill::FactorGraph` (the external dependency), and `AppComponent::Prediction`. `Team` finishes off the analysis of `models`; this is once again an `ActiveRecord` object and depends on `ActiveRecord::Base`. Finally, `views` does not contain any Ruby code.

Is that it? No refactoring opportunities? Not quite. Let's look at `Predictor` and `Prediction` again.

[./components/app\\_component/app/models/app\\_component/prediction.rb](#)

[Click here to view code image](#)

```
1 module AppComponent
2   class Prediction
3     attr_reader :first_team, :second_team, :winner
4
5     def initialize(first_team, second_team, winner)
6       @first_team = first_team
7       @second_team = second_team
8       @winner = winner
9     end
10    end
11  end
```

[./components/app\\_component/app/models/app\\_component/predictor.rb](#)

[Click here to view code image](#)

```
1 module AppComponent
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7           rating: [
8             Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
9           ]
10          }
11        memo
12      end
13    end
14
15    def learn(games)
16      games.each do |game|
17        first_team_rating =
```

```

18     @teams_lookup[game.first_team_id][:rating]
19     second_team_rating =
20         @teams_lookup[game.second_team_id][:rating]
21     game_result = game.winning_team == 1 ?
22         [first_team_rating, second_team_rating] :
23         [second_team_rating, first_team_rating]
24     Saulabs::TrueSkill::FactorGraph.
25         new(game_result, [1, 2]).update_skills
26     end
27 end
28
29 def predict(first_team, second_team)
30   team1 = @teams_lookup[first_team.id][:team]
31   team2 = @teams_lookup[second_team.id][:team]
32   winner = higher_mean_team(first_team, second_team) ?
33       team1 : team2
34   AppComponent::Prediction.new(team1, team2, winner)
35 end
36
37 def higher_mean_team(first_team, second_team)
38   @teams_lookup[first_team.id][:rating].first.mean >
39       @teams_lookup[second_team.id][:rating].first.mean
40 end
41 end
42 end

```

Maybe you noticed something while looking at `Prediction` and `Predictor`. For the two together, the following is true:

- `Predictor` and `Prediction` *together* are self-contained if one disregards the external dependency on `trueskill`. Granted teams and games are present in these files, but only as parameters that are passed in. `Predictor` does not know about the classes `Game` and `Team`. Also, for teams, all `Predictor` cares about (i.e., methods it calls) is `id`. For games, all it cares about is which teams played (`first_team_id`, `second_team_id`) and which team won (`winning_team`).
- When one does take a closer look at the dependency on `trueskill`, it becomes clear that no other part of the system depends directly on it.

If we now switch from the bottom-up approach to a top-down approach and think about the contribution of `Predictor` and `Prediction` to the application as a whole, everything lines up: `Prediction` stands out from the two other visible pieces of the app. Team and game management look very similar and do something very similar: They manage models—they do “model CRUD.” The predictor is quite different from a UI perspective, but also from a feature perspective. There is no CRUD in the predictor part. Also, in a way, prediction is at the core of what the Sportsball app is all about. This “UI-based” argument is very much in line with our understanding of the role of prediction in the app and the observation of the prediction code.

Let’s go ahead and extract a `Predictor`.

## 4.2 Refactoring: Extract Domain Gem—**Predictor**

The refactoring starts with the creation of the gem that will contain our newest component, Predictor.

### Create Predictor gem. Execute in . /

[Click here to view code image](#)

```
$ bundle gem predictor --no-bin --no-coc --no-ext \
    --no-mit --test=rspec
$ mv predictor/ components/
$ cd components/predictor
$ rm -rf .git*
```

The preceding commands create a new gem (check out all the options via `bundle --help gem`), moves it into the components directory, and—very important—deletes the newly created git root inside of the gem’s folder. The last step prevents us from having to deal with git submodules, which are completely unnecessary for CBRA.

---

### Why Git Submodules Are Not Needed

Git submodules are necessary for component-based application as much as they are *necessary* for any other form of application. No more. Submodules are not necessary. I am saying this so explicitly because I am commonly met with resistance regarding this point. One of the reasons for this is that gems and engines are generated with git files by default. This is, of course—and this is another reason for resistance—because gems and engines are *usually* used for distributing and sharing of code. As this is not the case for component-based Rails, these arguments don’t hold.

That said, similarly, submodules can also be used for the same effect for which they are used in other applications. For this use, it makes sense to cut submodule boundaries along component boundaries. As these submodules will be self-contained in that they contain all their tests, one could argue that they make for very good submodules.

In the end, I have not seen scenarios in which submodules would create advantages. I have only ever seen them as an unnecessary complication. If this kind of sharing is deemed necessary, I would recommend a private gem server over sharing via submodules. However, as I am a fan of the monorepo approach, I generally try to create situations that don’t require either.

---

### Boiling the Gem Down to Necessary Parts

If a `bin` folder was created, it can safely be deleted. Similarly, in the root of the gem, there are several files that can be deleted, including `.gitignore`, `.travis.yml`, and `CODE_OF_CONDUCT.md`. If needed, those files live in the root of the application.

I typically delete the `version.rb` that is created in gems by default. There is

not much benefit to having that file around—the version of the gem can just be hard-coded in the gem’s `gemspec`, as the version of the gem does not change while the gem is used as a component. To make this change, also remove the `requires` of this file from `predictor.gemspec` and `lib/predictor.rb`.

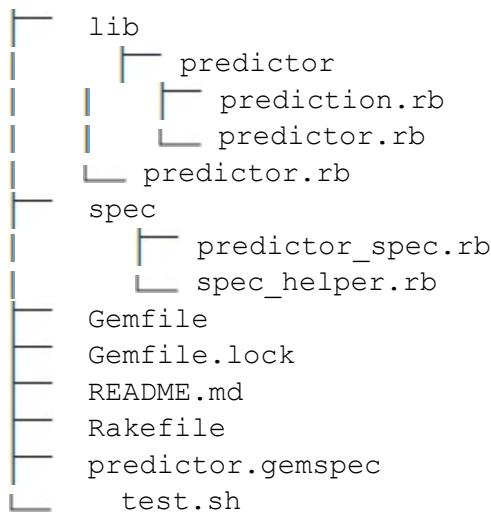
---

Next, `predictor.rb` and `prediction.rb` should be moved into the new gem’s `lib` folder—not into its root, but under `predictor`. Copy `test.sh` from the root of the `AppComponent` component to the root of `Predictor`. Finally, move `predictor_spec.rb`, which we take out of `components/app_component/spec/models/app_component/` and move into `components/predictor/spec/`.

That should result in the following folder and file structure:

### Structure of the Predictor gem

[Click here to view code image](#)



What follows is the process of fixing all the specs of the application, the most efficient approach to which is “from the inside out”:

1. Fix all specs in the new component.
2. Fix all specs of the component that directly depend on the previous component that was fixed.
3. Repeat step 2 until all components have been fixed.
4. Fix all specs in the container application.

This process ensures that at every step the component that is under review can rely on the tests of the lower layers to be working.

---

## Separation of Concerns Reflected in Broken Tests

As you do more of these refactorings, observe how many iterations of step 3 need fixing. Ideally, only the components *directly dependent* on the new component need to change. If their functionality is separated properly from other components using them, the test suites of those other components should be unaffected by the

refactoring.

If you do observe failures in not-directly-dependent components, check whether the components are truly testing their own contributions and not those from other components.

---

As you may recall from [Section 2.1](#), `predictor.gemspec` needs to be valid. For newer versions of `bundler`, this means that all “FIXME” and “TODO” entries need to be either removed or replaced with something meaningful.

Before we can run the first specs, we also need to fix the `test.sh` script that was copied over from the `AppComponent` component. We change its output to have the correct name and remove everything that is concerned with migrations. The resulting file should look as follows:

**`./components/predictor/test.sh`**

[Click here to view code image](#)

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running predictor gem specs"
6 #bundle install | grep Installing
7 bundle exec rspec spec
8 exit_code+=$?
9
10 exit $exit_code
```

The rest of this section follows in close detail every execution of `test.sh` for the `Predictor` gem until we finally fix the component. For every run of `test.sh`, I recommend only fixing the error indicated by the run’s output to ensure that every change is merited by a failing test.

**First run of `test.sh` for Predictor (some results omitted). Execute in**

**`./components/predictor`**

[Click here to view code image](#)

```
$ test.sh
*** Running predictor gem specs
```

...

```
sportsball/components/predictor/spec/predictor_spec.rb:1:in \
`<top (required)>': uninitialized constant AppComponent (NameError)
```

The problem is that `predictor_spec.rb` is still referring to the `AppComponent::` namespace.

**`./components/predictor/spec/predictor_spec.rb`**

[Click here to view code image](#)

```
1 RSpec.describe AppComponent::Predictor do
2   before do
```

```

3   @team1 = create_team name: "A"
4   @team2 = create_team name: "B"
5
6   @predictor = AppComponent::Predictor.new([@team1, @team2])
7 end
8
9 it "predicts teams that have won in the past to win
10    in the future" do
11   game = create_game first_team: @team1, second_team: @team2,
12     winning_team: 1
13   @predictor.learn([game])
14
15   prediction = @predictor.predict(@team2, @team1)
16   expect(prediction.winner).to eq @team1
17
18   prediction = @predictor.predict(@team1, @team2)
19   expect(prediction.winner).to eq @team1
20 end
21
22 it "changes predictions based on games learned" do
23   game1 = create_game first_team: @team1, second_team: @team2,
24     winning_team: 1
25   game2 = create_game first_team: @team1, second_team: @team2,
26     winning_team: 2
27   game3 = create_game first_team: @team1, second_team: @team2,
28     winning_team: 2
29   @predictor.learn([game1, game2, game3])
30
31   prediction = @predictor.predict(@team1, @team2)
32   expect(prediction.winner).to eq @team2
33 end
34
35 it "behaves funny when teams are equally strong" do
36   prediction = @predictor.predict(@team1, @team2)
37   expect(prediction).to be_an AppComponent::Prediction
38   expect(prediction.first_team).to eq @team1
39   expect(prediction.second_team).to eq @team2
40   expect(prediction.winner).to eq @team2
41
42   prediction = @predictor.predict(@team2, @team1)
43   expect(prediction).to be_an AppComponent::Prediction
44   expect(prediction.first_team).to eq @team2
45   expect(prediction.second_team).to eq @team1
46   expect(prediction.winner).to eq @team1
47 end
48 end

```

The solution is to replace the `AppComponent::` namespace with the `predictor` namespace to create both `Predictor::Predictor` and `Predictor::Prediction`.

Run `test.sh` again and we get the following error:

**Second run of `test.sh` for Predictor (some results omitted). Execute in `./components/predictor`**

[Click here to view code image](#)

```
$ test.sh
*** Running predictor gem specs

. . .

sportsball/components/predictor/spec/predictor_spec.rb:1:in \
`<top (required)>': uninitialized constant Predictor::Predictor \
(NameError)
```

When we open `predictor.rb` (and `prediction.rb`), we see that it too uses the old `AppComponent::` namespace, which should again be renamed to `Predictor::`.

**`./components/predictor/lib/predictor/predictor.rb`**

[Click here to view code image](#)

```
1 module Predictor
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7
7           rating: [
8             Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
9           ]
10          }
11        memo
12      end
13    end
14  end
15
16  def learn(games)
17    games.each do |game|
18      first_team_rating = @teams_lookup[game.
19        first_team_id][:rating]
20      second_team_rating = @teams_lookup[
21        game.second_team_id][:rating]
22      game_result = game.winning_team == 1 ?
23        [first_team_rating, second_team_rating] :
24        [second_team_rating, first_team_rating]
25      Saulabs::TrueSkill::FactorGraph.
26        new(game_result, [1, 2]).update_skills
27    end
28  end
29
30  def predict(first_team, second_team)
31    team1 = @teams_lookup[first_team.id][:team]
32    team2 = @teams_lookup[second_team.id][:team]
33    winner = higher_mean_team(first_team, second_team) ?
34      team1 : team2
35    ::Predictor::Prediction.new(team1, team2, winner)
36  end
37
38  def higher_mean_team(first_team, second_team)
39    @teams_lookup[first_team.id][:rating].first.mean >
40      @teams_lookup[second_team.id][:rating].first.mean
41  end
```

```
42     end
43   end
44 end
```

**./components/predictor/lib/predictor/prediction.rb**

[Click here to view code image](#)

```
1 module Predictor
2   class Prediction
3     attr_reader :first_team, :second_team, :winner
4
5     def initialize(first_team, second_team, winner)
6       @first_team = first_team
7       @second_team = second_team
8       @winner = winner
9     end
10    end
11  end
```

While it may be unusual to have the component's module name also be the name of one of its classes, the code is perfectly viable. However, as discussed in [Section 2.1](#), having the same name for two different things will make the refactoring of either of these names harder, as it requires more work in the creation of renaming logic to separate the two. While negative, the effect is likely to not be severe because the module Predictor is quite small. Nonetheless, if one wanted to prevent this, one could go with something like Predicting or Predictions instead of Predictor for the module name. However, as I can't quite warm up to these alternatives, I am sticking with this choice.

If we fix the namespace in the preceding files and rerun the specs, the error is the same. The reason for this is that in gems, by default, there is no such thing as autoloading. We have not yet ensured that predictor.rb is actually loaded by the gem. To fix this, add the needed require to lib/predictor.rb:

**./components/predictor/lib/predictor.rb**

[Click here to view code image](#)

```
1 module Predictor
2   require "predictor/predictor"
3   require "predictor/prediction"
4 end
```

With this change, the specs actually run! They all fail—but this is nonetheless a huge step. With just a few changes, the new gem starts to come to life!

Currently, create\_team missing is the single reason for the failure of all specs.

**Third run of test.sh for Predictor. Execute in ./components/predictor**

[Click here to view code image](#)

```
$ test.sh
*** Running predictor gem specs
```

```
Predictor::Predictor
predicts teams that have won in the past to win in the future \
(FAILED - 1)
```

```
changes predictions based on games learned (FAILED - 2)
behaves funny when teams are equally strong (FAILED - 3)
```

Failures:

```
1) Predictor::Predictor predicts teams that have won in the past \
   to win in the future
Failure/Error: @team1 = create_team name: "A"
NoMethodError:
  undefined method `create_team' for \
  #<RSpec::ExampleGroups::PredictorPredictor:0x007fed8581e9b0>
  # ./spec/predictor_spec.rb:3:in `block (2 levels) in
  # <top (required)>'

# ...

Finished in 0.00099 seconds (files took 0.69315 seconds to load)
3 examples, 3 failures
```

Remember that `create_team` is the object creation method we created in [Section 3.1](#) which creates a `Team` for tests. Look further down in the spec and notice that the other object creation method, `create_game`, is also used in this spec. Does this mean `Team` and `Game` are needed in this gem? That would mean that all the models would be sucked into our gem—not really what we had in mind when trying to extract it.

What happened? Is this refactoring a bust? Do we have to roll it back? Quite the opposite: These failures actually show the power of component-based refactorings! We have just found a place where our code was unnecessarily knowledgeable about another piece of code. We have found an *entanglement* and we can now set out to break it up.

Look at the implementation and the specs of `Predictor` once again. As we discussed before, the implementation *is not dependent* on `Team` or `Game` (the classes). Yet, through `create_team` (and `create_game`) the specs *are dependent* on those two classes. If the implementation doesn't need it, the specs should be able to live without it!

There are a couple of ways to remove this dependency. As we discussed, all the predictor cares about regarding teams are their IDs, and for games, all it cares about is which teams played and which one won. When we state these requirements, we are no longer using the class names `Game` and `Test` to describe what is going on. And, in fact, in our tests, we can use any object as a stand-in for `Team` that has an `id` method. Similarly, any object responding to `first_team_id`, `second_team_id`, and `winning_team` can stand in for `Game`.

We use `OpenStruct` (<http://ruby-doc.org/stdlib-2.2.2/libdoc/ostruct/rdoc/OpenStruct.html>) from the Ruby Standard Library to create a stand-in object for our tests. This stand-in responds to `id`, which allows us to use it instead of the `Team` model we used previously. The same pattern is used to get rid of the dependency on `Game`.

`./components/predictor/spec/predictor_spec.rb`

[Click here to view code image](#)

```
1 require_relative "spec_helper.rb"
2
3 RSpec.describe Predictor::Predictor do
4   before do
5     @team1 = OpenStruct.new(id: 6)
6     @team2 = OpenStruct.new(id: 7)
7
8     @predictor = Predictor::Predictor.new([\@team1, \@team2])
9   end
10
11  it "predicts teams that have won in the past to
12    win in the future" do
13    game = OpenStruct.new(
14      first_team_id: @team1.id, second_team_id: @team2.id,
15      winning_team: 1)
16    @predictor.learn([game])
17
18    prediction = @predictor.predict(@team2, @team1)
19    expect(prediction.winner).to eq @team1
20
21    prediction = @predictor.predict(@team1, @team2)
22    expect(prediction.winner).to eq @team1
23  end
24
25  it "changes predictions based on games learned" do
26    game1 = OpenStruct.new(first_team_id: @team1.id,
27      second_team_id: @team2.id, winning_team: 1)
28    game2 = OpenStruct.new(first_team_id: @team1.id,
29      second_team_id: @team2.id, winning_team: 2)
30    game3 = OpenStruct.new(first_team_id: @team1.id,
31      second_team_id: @team2.id, winning_team: 2)
32    @predictor.learn([game1, game2, game3])
33
34    prediction = @predictor.predict(@team1, @team2)
35    expect(prediction.winner).to eq @team2
36  end
37
38  it "behaves funny when teams are equally strong" do
39    prediction = @predictor.predict(@team1, @team2)
40    expect(prediction.first_team).to eq @team1
41    expect(prediction.second_team).to eq @team2
42    expect(prediction.winner).to eq @team2
43
44    prediction = @predictor.predict(@team2, @team1)
45    expect(prediction.first_team).to eq @team2
46    expect(prediction.second_team).to eq @team1
47    expect(prediction.winner).to eq @team1
48  end
49 end
```

While OpenStruct is in the Standard Library, it is not required and loaded by default. So, before we run test.sh again, we add a require to ostruct, which provides OpenStruct to spec\_helper.rb:

## **./components/predictor/spec/spec\_helper.rb (partial)**

[Click here to view code image](#)

```
1 require "bundler/setup"
2 require "predictor"
3 require "ostruct"
```

The next run of `test.sh` shows that we are getting past the problem of the unrelated models: Team and Game. We do, however, face a different unknown constant, namely, Saulabs. When Ruby cannot find a constant in the lexical scope of the code, it will report it with the fully qualified name of that scope. That is why the errors all read uninitialized constant Predictor::Predictor::Saulabs. What the interpreter obviously cannot know is that Saulabs is in fact a top-level name created by the `trueskill` gem.

**Fourth run of `test.sh` for Predictor (some results omitted). Execute in**

## **./components/predictor**

[Click here to view code image](#)

```
$ test.sh
*** Running predictor gem specs

Predictor::Predictor
predicts teams that have won in the past to win in the future \
(FAILED - 1)
changes predictions based on games learned (FAILED - 2)
behaves funny when teams are equally strong (FAILED - 3)
```

Failures:

```
1) Predictor::Predictor predicts teams that have won in the past \
to win in the future
Failure/Error: @predictor = \
Predictor::Predictor.new([@team1, @team2])
NameError:
uninitialized constant Predictor::Predictor::Saulabs
```

. . .

```
Finished in 0.00102 seconds (files took 0.16532 seconds to load)
3 examples, 3 failures
```

To add `trueskill` as a dependency, we must add it to the `gemspec` to call it out as a runtime dependency. We must add it to the `Gemfile` because the particular version of the gem we are using is provided by a git repository (check back to [Section 2.3](#) to see why that is). Last, we must ensure that the gem is loaded by adding a `require` to our `gemspec`.

## **./components/predictor/predictor.gemspec**

[Click here to view code image](#)

```
1 # coding: utf-8
2 lib = File.expand_path("../lib", __FILE__)
3 $LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)
4
```

```

5 Gem::Specification.new do |spec|
6   spec.name          = "predictor"
7   spec.version       = "0.1.0"
8   spec.authors       = ["Stephan Hagemann"]
9   spec.email         = ["stephan.hagemann@gmail.com"]
10
11  spec.summary       = %q{Prediction Core}
12
13  spec.files = Dir["{lib}/**/*", "Rakefile", "README.md"]
14  spec.require_paths = ["lib"]
15
16  spec.add_dependency "trueskill"
17
18  spec.add_development_dependency "bundler"
19  spec.add_development_dependency "rake"
20  spec.add_development_dependency "rspec"
21 end

```

## **./components/predictor/Gemfile**

[Click here to view code image](#)

```

1 source "https://rubygems.org"
2
3 # Specify your gems dependencies in predictor.gemspec
4 gemspec
5
6 gem "trueskill",
7   git: "https://github.com/benjaminleesmith/trueskill",
8   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"

```

## **./components/predictor/lib/predictor.rb**

[Click here to view code image](#)

```

1 require "saulabs/trueskill"
2
3 module Predictor
4   require "predictor/predictor"
5   require "predictor/prediction"
6 end

```

Run test.sh again—and voilà—the Predictor tests are passing.

## **Fifth run of test.sh for Predictor. Execute in ./components/predictor**

[Click here to view code image](#)

```

$ test.sh
*** Running predictor gem specs

...

```

```

Finished in 0.00269 seconds (files took 0.12203 seconds to load)
3 examples, 0 failures

```

## 4.2.1 Fixing Inside-Out: Making AppComponent Work Again

As per our routine described, we should now turn to the first component that directly requires (or more accurately, *should* require) Predictor. Since we extracted Predictor from AppComponent, AppComponent will need to depend on Predictor to continue working. Let's switch into the folder of AppComponent and run test.sh once again.

**First run of test.sh for AppComponent (some results omitted). Execute in  
./components/app\_component**

[Click here to view code image](#)

```
$ test.sh
*** Running app component engine specs
..F.....F.....
```

Failures:

- 1) AppComponent::PredictionsController POST create assigns a prediction as @prediction  
Failure/Error: post :create,  
NameError:  
 uninitialized constant AppComponent::Predictor  
# ./app/controllers/app\_component/predictions\_controller.rb:8
# :in `create'
# ...
  
- 2) the prediction process get a new prediction  
Failure/Error: click\_button 'What is it going to be'  
NameError:  
 uninitialized constant AppComponent::Predictor  
# ./app/controllers/app\_component/predictions\_controller.rb:8
# in `create'
# ...

```
Finished in 1.64 seconds (files took 1.68 seconds to load)
69 examples, 2 failures
```

I hope you expected this issue at this point. Nothing in AppComponent has been changed so far to make it aware of the newly created Predictor:: Predictor, and consequently it still refers to the old namespaces for Predictor and Prediction. A good way to quickly find all the bad references is to search and replace every occurrence of AppComponent:: Predict with Predictor::Predict within the folder structure of the AppComponent gem.

**Second run of test.sh for AppComponent (some results omitted). Execute in  
./components/app\_component**

[Click here to view code image](#)

```
$ test.sh
*** Running app component engine specs
.....F.....F..
```

Failures:

```
1) AppComponent::PredictionsController POST create assigns a \
   prediction as @prediction
Failure/Error: post :create,
NameError:
  uninitialized constant AppComponent::PredictionsController::\
  Predictor
# ./app/controllers/app_component/predictions_controller.rb:8:\
#   in `create'
#...

2) the prediction process get a new prediction
Failure/Error: click_button 'What is it going to be'
NameError:
  uninitialized constant AppComponent::PredictionsController::\
  Predictor
# ./app/controllers/app_component/predictions_controller.rb:8:\
#   in `create'
#...
```

Finished in 1.96 seconds (files took 3.05 seconds to load)  
69 examples, 2 failures

Again, a hopefully unsurprising error, which is structurally the same as the Saulabs constant that was missing earlier in Predictor. Just like before, the answer includes changes to the gemspec, the Gemfile, and the requirement of the gem.

**./components/app\_component/app\_component.gemspec**

[Click here to view code image](#)

```
1 $:.push File.expand_path("../lib", __FILE__)
2
3 # Maintain your gems version:
4 require "app_component/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name      = "app_component"
9   s.version   = AppComponent::VERSION
10  s.authors   = ["Stephan Hagemann"]
11  s.email     = ["stephan.hagemann@gmail.com"]
12  s.homepage  = ""
13  s.summary   = "Summary of AppComponent."
14  s.description = "Description of AppComponent."
15  s.license   = "MIT"
16
17 s.files = Dir["{app,config,db,lib}/**/*",
18   "MIT-LICENSE", "Rakefile", "README.md"]
19
20 s.add_dependency "rails", "5.1.4"
21 s.add_dependency "slim-rails", "3.1.3"
22 s.add_dependency "jquery-rails", "4.3.1"
23 s.add_dependency "trueskill"
24 s.add_dependency "predictor"
25
```

```
26   s.add_development_dependency "sqlite3"
27   s.add_development_dependency "rspec-rails"
28   s.add_development_dependency "shoulda-matchers"
29   s.add_development_dependency "database_cleaner"
30   s.add_development_dependency "capybara"
31   s.add_development_dependency "rails-controller-testing"
32 end
```

## ./components/app\_component/Gemfile

[Click here to view code image](#)

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 path "../" do
6   gem "predictor"
7 end
8
9 gem "trueskill",
10 git: "https://github.com/benjaminleesmith/trueskill",
11 ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

## ./components/app\_component/lib/app\_component.rb

[Click here to view code image](#)

```
1 require "slim-rails"
2 require "jquery-rails"
3
4 module AppComponent
5   require "app_component/engine"
6   require "predictor"
7 end
```

Notice that the trueskill gem is no longer required from app\_component.gemspec and that the explicit requiring of saulabs/trueskill is gone from app\_component.rb. This is because AppComponent does not make use of trueskill directly and its use has now become an implementation detail of Predictor.

---

## The Danger of the Accidental Use of Public Gems

If the preceding changes make a test run result in uninitialized constant Predictor::Predictor, an entirely different reason might be to blame. Run bundle and look for the line starting Using predictor—how does it end?

- Using predictor 0.1.0 from source at components—this is right—the gem is being loaded from the local path components
- Using predictor (2.3.0)—this is wrong—no local path and the wrong version; we accidentally downloaded the predictor gem (<https://rubygems.org/gems/predictor>) from rubygems.

The predictor gem on rubygems has nothing to do with the component we

just created and does not define Predictor:: Predictor, hence the error message. This danger lurks for every component that has a name that is also the name of a gem published on rubygems.

---

## Preventing Accidental Use of Public Gems

A little hack can help prevent the accidental use of public gems over the components defined in our application.

It is important to recall that CBRA does not make use of the version number of gems to find compatible component versions. Instead, the entire app, Rails container plus engine components, is checked in to one version control repository. The path option used for referencing gems in gemfiles ensures that the local version of the gem is used. Because of this, we don't ever need to change the versions of components and can use it to prevent the problem of accidentally pulling in the code of a published gem with the same name.

1. Set the version of a CBRA component's gem to a very *special version*, say "0.3.14159265359". In this case, the minor and path part of the version number are the first digits of  $\pi$ . More importantly, the patch number is very long—published gems don't commonly have that kind of patch number (making a "version duplicate" highly unlikely). The version number can be the same for all components in the app. Making all version numbers the same should make human parsing easier.
2. Require every CBRA component with an explicit version number in the gemspecs of other components: spec.add\_dependency "predictor", "0.3.14159265359"

This way, even if something is wrong with the setup of the gem's load path in the Gemfile of any included component, instead of loading some random gem from rubygems, bundling will fail with a useful error message:

[Click here to view code image](#)

```
Bundler could not find compatible versions for gem "predictor":
```

```
In Gemfile:
```

```
app_component (>= 0) ruby depends on
  predictor (= 0.3.14159265359) ruby
```

```
Could not find gem 'predictor (= 0.3.14159265359) ruby',
which is required by gem 'app_component (>= 0) ruby',
in any of the sources.
```

It may seem counterintuitive, but this trick is not in conflict with versioning schemes like semantic versioning.<sup>a</sup> While part of a component-based Rails app, a component's gem is not shared with other applications, so no one will be relying on its version. Should the component be extracted from the app and published (publicly or privately), the new maintainers of the gem could simply adopt semantic versioning with its initial publication. No harm done.

---

Run test.sh again:

### **Third run of test.sh for AppComponent. Execute in ./components/app\_component**

[Click here to view code image](#)

```
$ test.sh
*** Running app component engine specs
.....
Finished in 1.69 seconds (files took 1.79 seconds to load)
69 examples, 0 failures
```

Randomized with seed 35251

It is a good sign that fixing AppComponent took much less effort than fixing Predictor. There were also no unexpected failures; the two that we had to fix were due to the renaming of the namespace of the two prediction classes and due to the creation of a new gem.

#### **4.2.2 Last Step: Ensure That the App Works**

The last step is to cd into the directory of the main application and run test.sh there.

### **First run of test.sh for the container app (Some results omitted). Execute in ./**

[Click here to view code image](#)

```
$ test.sh
*** Running container app specs
....
```

Randomized with seed 11686

```
.....
Finished in 0.65533 seconds (files took 2.55 seconds to load)
4 examples, 0 failures
```

Because we worked our way from the inside of our change to the outside, the Rails application is relying on the two components that are already fixed. Also, because the abstractions are working and the distribution of work between the AppComponent and the Predictor component is transparent to the main application, no changes are necessary to make the container app work again—it never broke!

---

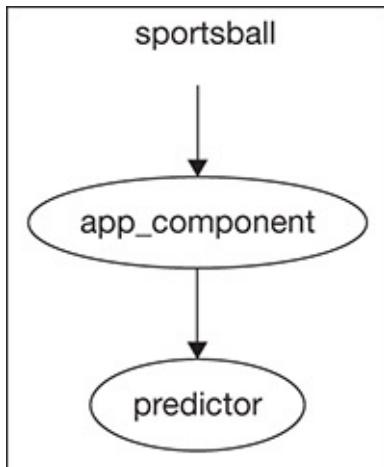
## Splitting Off the Entire Prediction Vertical

A good argument can be made for not just extracting the domain gem as we did in this section, but rather splitting off the entire prediction vertical, including its controllers and views. That refactoring, however, takes an additional step, which we will discuss in an upcoming section.

---

With the extraction of Predictor, we now have three distinct parts to our application. Using the cobradeps (<https://rubygems.org/gems/cobradeps>) gem, we can quickly

create the diagram of the CBRA dependencies in the app. Its output is shown in [Figure 4.3](#).



**Figure 4.3.** Current state of dependencies within Sportsball

#### 4.2.3 Summary of Bottom-Up Component Extractions

Bottom-up component extractions work first and foremost with the code by analyzing it for parts that are more or less self-contained. These parts can be found by looking at the folder structure, the filename, class inheritance patterns, and the methods in classes. For a prospective new component, a new gem is created and the identified files and their tests are moved into the new gem. Tests won't pass because of incorrect naming of modules and missing external dependencies. Continuous test runs eventually lead to the situation where the tests run but are red because of loading or naming issues, and because of ties to content of the originating component (dare I say, dependencies). Once the new component's tests pass, we iterate our way up the dependency graph and fix every higher-level component until the container application is fixed.

The bottom-up component extraction approach typically leads to gems that are small in comparison to the entire application—maybe a couple of classes. They tend to encompass classes from the model layer (but not ActiveRecord models) and do not include controllers or views. These gems might have external dependencies, but often do not depend on Rails. Even if Rails seems to be a dependency, it can often be refactored away by removing a couple of uses of the Rails API. The extraction of ActiveRecord models is often more involved than that of non- ActiveRecord classes. However, there can be less entangled and mostly self-contained ActiveRecord objects that lend themselves to bottom-up extraction. In this case, instead of just a gem, we would create an engine to properly load Rails for testing.

In comparison to bottom-up extraction, top-down extraction tends to move large parts of an application into components. We will discuss it next to understand the remainder of the structure of Sportsball. From that understanding, the remaining component extractions will become clear and we can start slicing and dicing the remainder of the application into meaningful components.

## 4.3 Determining What to Extract: Top-Down

Where bottom-up component extraction raises the question of what certain classes mean to the domain somewhere in the middle of its process, top-down extraction *starts* with that question. In an even more generic form, the question is: What are the parts of this application and how are they structurally related?

One of the best ways of addressing this question is to start with the different users of the system. Are there multiple types? Who are they? How do they differ? What are the different things they do within the application?

Different user types and the different actions they perform within the system are good indicators of seams within the application. In the case of Sportsball, we might imagine that there are two separate administration jobs: teams admin and games admin—one manages the teams, the other the games played.

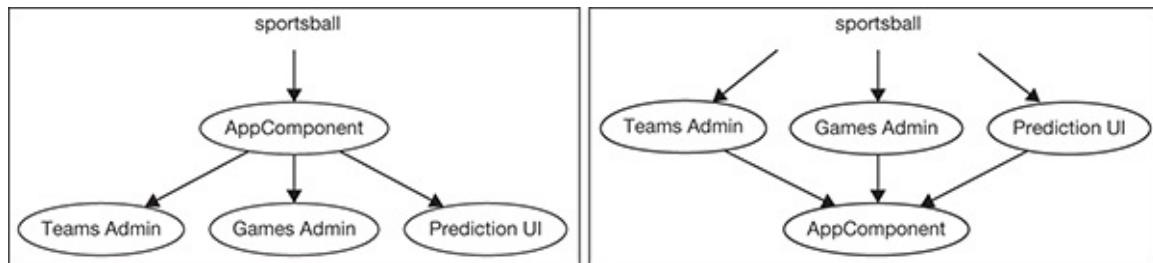
Why not just one admin, you might ask? Think about the different situations that trigger the entry of data. For games, during the season, this is a recurring task every week, maybe even more often. Whenever games are played, if Sportsball is updated as soon as the results are available, predictions will, we hope, benefit from the newly available data. Teams, on the other hand, will not need a lot of managing. In fact, in many leagues, the creation of a new team or the abandonment of an existing one is a rare event—one could consider not storing teams in a database at all, but rather as dictionaries (i.e., hashes) defined in code. Since we have chosen to store teams in a database table and we created a UI for it, we might as well take these differences in management patterns as sufficient evidence that the admin portion of the app can reasonably be considered to consist of two separate admin sections.

It follows from this discussion that it is possible to consider the prediction portion of the application as a separate component as well. Its use is very distinct from each of the two admins: It is used by someone interested in predicting the outcome of a game that has not yet been played. This could happen at any point between games within the season or between seasons.

After accepting this tentative list of components (teams admin, games admin, and prediction interface), we need to map the dependencies between components in order to fully understand the new structure of the app.

### 4.3.1 Unclear Dependency Direction

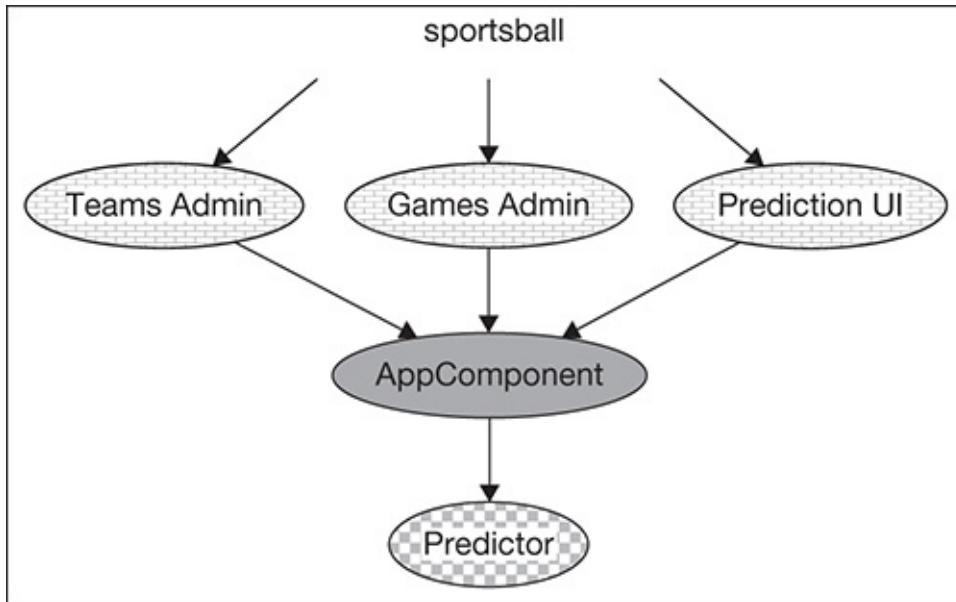
The first observation regarding dependencies between components is that what started as a conscious decision for Predictor is now an open question: Should the new engines be dependent upon AppComponent or should AppComponent depend on them (see [Figure 4.4](#))? In the case of Predictor, we determined that part of the code could “live alone” and did the refactoring response to that. Now, if AppComponent were to depend on the new components, AppComponent would (only) mount the new components and the components would have to define everything they need (since they cannot depend on AppComponent). If the new components depend on AppComponent, the container application would need to mount the new components and they could continue using what is defined in AppComponent. For Predictor, we sought independence because the code showed that it didn’t need other parts of the application. Now we are looking at user interfaces that by nature depend on everything that is defined in the domain, that is, models and other stuff. Predictor extracted the “bottom” of the domain, whereas this refactoring is extracting the “top” of the domain.



**Figure 4.4.** What are the dependencies?

Based on what we are extracting, we will have the new components—teams admin, games admin, and prediction interface—depend on what we extracted them from so that they can continue to use the other parts of the application.

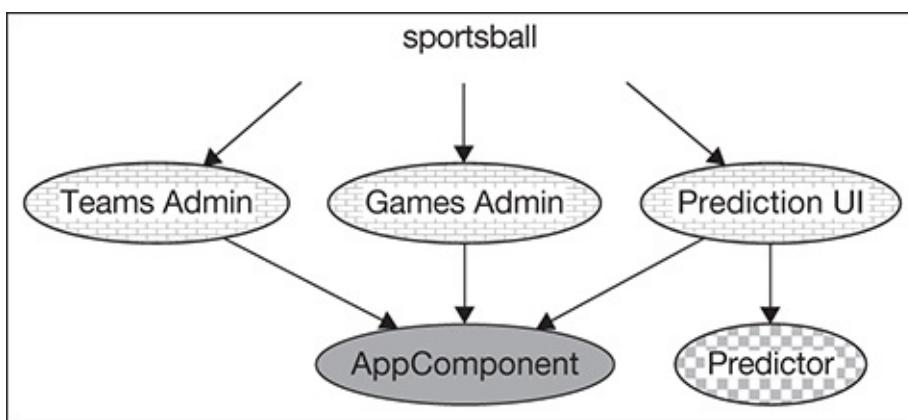
In [Figures 4.5 through 4.11](#), domain gems, which contain only classes belonging to the domain of the app, are denoted by the checkerboard fill (Predictor), UI engines (controllers, helpers, and views only) are denoted by the brick-pattern fill (TeamsAdmin, GamesAdmin, PredictionUI), and engines with mixed responsibilities have solid fill (AppComponent).



**Figure 4.5.** New components depending on AppComponent

#### 4.3.2 Not Everything Needs Predictor

As the reader may remember from the discussion of the Predictor component, the only part outside of Predictor that uses it is the PredictionController. Given that we are now intending to extract prediction UI as a component, AppComponent no longer needs to depend on Predictor. With this, we can change the dependency graph to the one depicted in [Figure 4.6](#), in which PredictionUI directly depends on Predictor.



**Figure 4.6.** Direct dependency of PredictionUI on Predictor

As small as this change may seem, conceptually it has a huge impact on the overall complexity of the application. Remember that before the extractions of the three components, AppComponent contained everything there was in this application. Now, by just splitting off three UI pieces, we have identified that most of the application, namely games admin, teams admin, and AppComponent, does not need Predictor at all. Predictor changing and potentially breaking something in one of these components is no longer a possibility.

### 4.3.3 Implications of the First Set of Extractions

What is included into these new components? We arrived at the component split through the definition of their UIs. As we already discussed, that means that controllers and views will naturally be able to move up into the respective engines. Any helpers should easily move up too—no complications here.

The ease of these decisions changes when we look at the models. It seems like all the new engines need all the models. Or do they? Can models move up into the new engines? If so, which ones? Let's list *who* needs *what*:

- Teams Admin: needs Team—it appears not to need Game
- Games Admin: needs Game and Team (the latter is needed to allow for game management if we don't just want to deal with team IDs and to ensure that valid team IDs are used)
- Prediction UI: needs both Game (all games, in fact, to allow for the prediction algorithm to learn) and Team for the selection of teams for which we are predicting an outcome

---

### Theoretically Possible Model to Component Refactorings

It is interesting to analyze how the model requirements guide the decision of which model should go into which new component.

Could any of the new components contain the models? Could the models be split up between components?

#### All Models in One Engine

It is trivial to prove that one option is to move both models into the same new component. Take the prediction interface for example. If both models were moved in there, they would be renamed to `PredictionUI::Game` and `PredictionUI::Team`. To make games admin and teams admin work, they both depend on the prediction interface. `AppComponent` depends on all three new components. After the appropriate refactorings, the app will work like before.

This argument works for all three new components in the same way. See options 1, 2, and 3 in [Table 4.1](#).

**Table 4.1** Which models can be defined in which engines?

---

	games	admin	teams	admin	prediction	UI	works?	dependencies
Needs	Team, Game	Team		Team, Game	N/A	N/A		
Option 1	Team, Game				yes	teams admin, prediction		UI -> games admin

Option 2	Team, Game	yes	games admin, prediction UI -> teams admin
Option 3	Team, Game	yes	teams admin, games admin -> prediction UI
Option 4	Team	Game	no N/A
Option 5	Team	Game	no N/A
Option 6	Game	Team	yes prediction UI -> games admin -> teams admin
Option 7	Team	Game	yes games admin -> prediction UI -> teams admin
Option 8	Game	Team	no N/A
Option 9	Game	Team	no N/A

### Is “All Models in One Engine” Forced?

So, it is *possible* for the models to be defined in the same engine, but is it *necessary*? Let’s start again with prediction UI. If it were to define only one model, say Team, that would force games admin (the only other component that also needs Game) to define Game and for prediction UI to depend on it. In that case, however, games admin would need to depend on prediction UI for it to have access to Team. Trying to switch the two engines will lead to same problem. See options 5 and 8 in [Table 4.1](#).

A similar problem occurs if teams admin defines Game and Team is defined in either games admin or prediction UI. Teams admin needs to depend on the respective other component, and it in turn needs to depend on teams admin. See options 4 and 9 in [Table 4.1](#).

The one set of options left to analyze is for teams admin to define Team and for one of the other engines to define Game. That leads to options 6 and 7 in [Table 4.1](#) and it turns out that both of these options work. If games admin defines Game, it will depend on teams admin and in turn prediction UI will depend on it. Similarly, if prediction UI defines Game, it depends on teams admin and games admin depends

on it.

Note that the transitive dependency of prediction UI on teams admin (or games admin on teams admin) is used here only for clarity of the analysis. In both cases the “top” engine actually *directly* uses a model from an engine it depends on *transitively*. This direct dependency indicates that the other engines are not merely “implementation details” to itself. We see this reflected in the statement that prediction UI and games admin “care” about whether the full name of Team is Teams::Team or something else, because they refer to this constant in “their” code.

In conclusion, 5 out of 9 options of defining the models in the different engines are possible. Three of these options leave the models in the same engine, and two split Team and Game into different engines.

If we apply the mantra “what can be separate should be separate,” we should favor option 6 or 7, as those are the ones that create smaller components and prevent a “supercomponent” that knows all models.

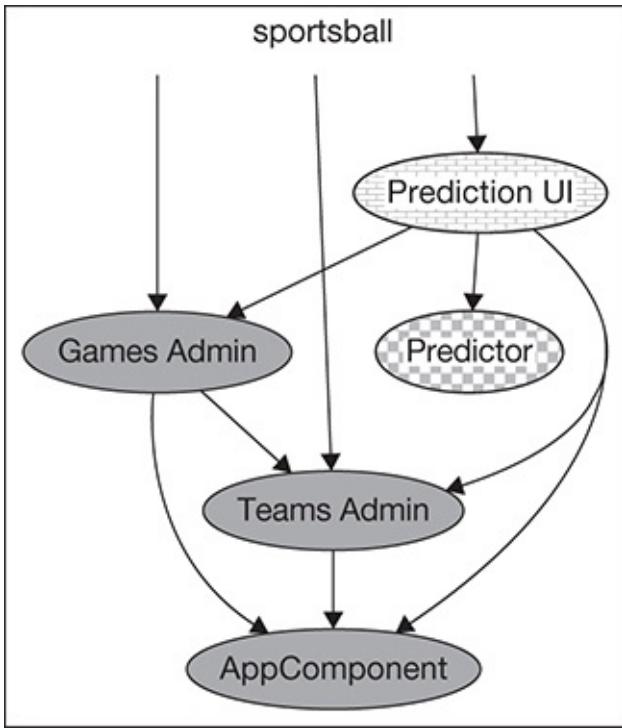
Finally, is there a way to decide whether option 6 or option 7 is *better*? Look at the full model names for an answer:

- Option 6 (games admin defines Game: TeamsAdmin::Team and GamesAdmin::Game)
- Option 7 (prediction UI defines Game: TeamsAdmin::Team and PredictionUI::Game)

Option 6 exhibits consistency whereas option 7 does not. With option 6, the admin components define the models and both models are namespaced in a component that contains their own name!

---

You may have intuitively gotten that the best way to split the models is for teams admin to define Team and for games admin to define Game, or you may have read through the theoretical analysis. Either way, in this case, there is a *best way* to define the models and construct the dependencies. It is shown in [Figure 4.7](#).



**Figure 4.7.** The *best way* to split components in this app? Really!?

### Extractions Lead to Extractions

Creating components in applications is a slippery slope (of a good kind). It is very common for the extraction of one component to trigger the wish or need to extract another component. Go nuts! Follow your instincts and extract what seems right! It *never* hurts to extract too much; it will always teach something, and it is *always* reversible. In fact, it is *significantly* easier to (re)combine components than it is to extract them.

#### 4.3.4 Problem with the Current Solution

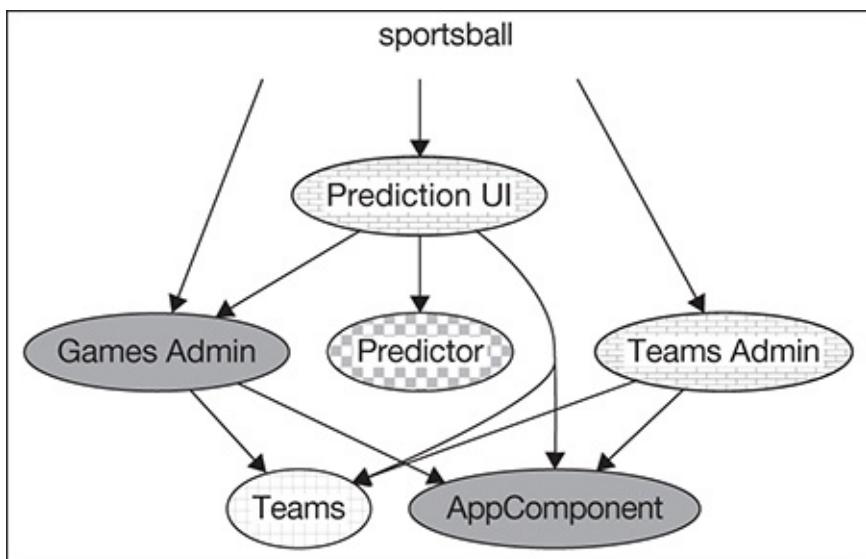
You probably never actually believed that the current version of the app was the best way it could be split up!

Here is one piece of criticism that I hope is obvious by now: `PredictionUI` depends on all of games admin and all of teams admin, but it does not need access to their controllers and views. Remember, if we don't *need* a dependency, we probably *should not* have it. Also, prior to the current refactoring, games admin and teams admin were denoted with brick fill in the dependency diagram, because they initially were UI engines. Now they have solid fill as they handle more things. While that is not inherently bad, it shows that the last step introduced some entanglement that previous steps did not include.

#### 4.3.5 Reducing Needed Dependencies

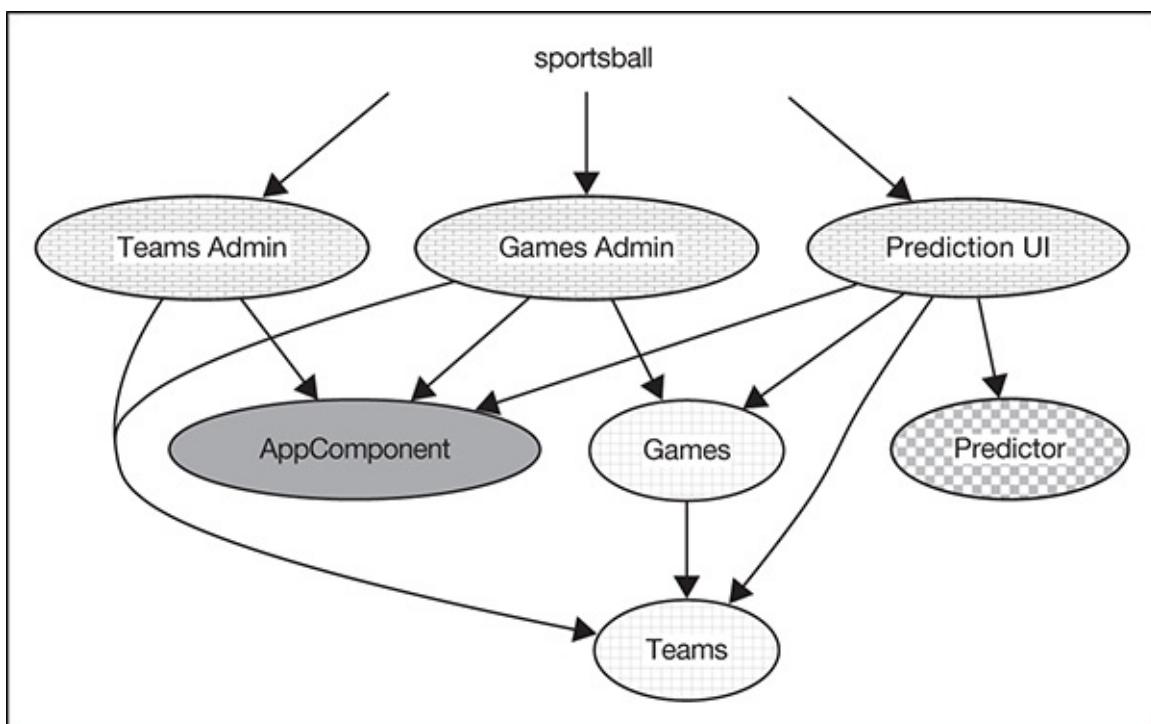
The similarity between the too-large dependency of GamesAdmin on TeamsAdmin and PredictionUI on both of the admins is that in all cases, what is needed by the higher-level component are only the ActiveRecord models Team and Game. If there were components for each of these models, the admin and UI components would be able to depend on exactly what they needed.

If, for example, we pull Team down out of TeamsAdmin into, say, Teams, then GamesAdmin and PredictionUI can both depend on Teams. As a side effect, TeamsAdmin no longer mixes concerns and turns back into a UI engine and can be depicted in with brick fill in [Figure 4.8](#), which shows this change. Teams is a model engine, which we denote with the grid-patterned fill.



**Figure 4.8.** Teams extracted out of TeamsAdmin

Similarly, a Games component can be extracted out of GamesAdmin as another model engine, which leads to the graph seen in [Figure 4.9](#).

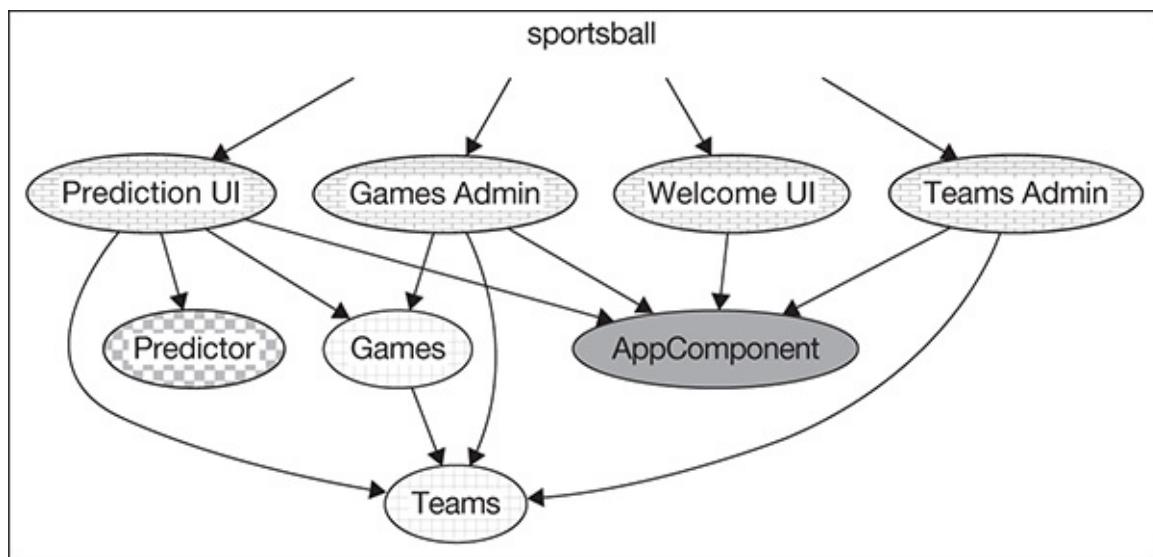


**Figure 4.9.** Games extracted out of GamesAdmin

At this point, all the new components have in common that they are all very singular in nature; Teams and Games define only one model each. Predictor defines two classes, but two that are extremely closely related. All three UI engines, TeamsAdmin, GamesAdmin, and PredictionUI, support one use case. What does AppComponent do?

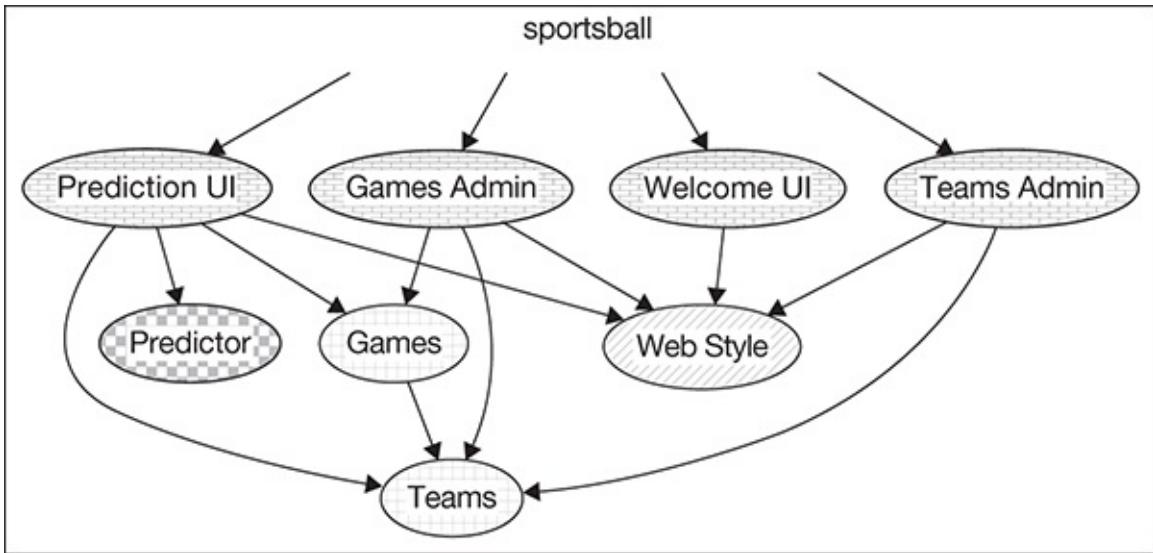
The name AppComponent never had a meaning in our application. It is much too broad a name, and the component was a catch-all for what we did not want to put into the main application. The refactorings discussed in this chapter greatly reduce the contents remaining in AppComponent. What is left in it now? Really only two things: some assets, used by all of the UI engines, and one more controller, the WelcomeController.

WelcomeController is easy to overlook as a controller, because it is not really a use case; it is the landing page of the app, which for Sportsball does not do much of anything. If we, just for kicks, pull this controller and its views up into a WelcomeUI component, we get the dependency graph of [Figure 4.10](#).



**Figure 4.10.** Extracted WelcomeUI

Now, finally, AppComponent is a component with a singular contribution as well. It provides the app's public UI assets: images, CSS, and JavaScript. Denoting it as a UI asset engine and filling it in with striped fill, there is only one thing left to do: Get rid of that name, "AppComponent." Let's refactor this engine to be called WebStyle to denote that it is the engine that gives the app its looks. We finally end up with the dependency graph of [Figure 4.11](#).



**Figure 4.11.** WebStyle instead of AppComponent

At this stage, every component in Sportsball does only one thing. Every component has minimal dependencies. Every component has a decent name—in theory, that is. We still need to discuss all the refactorings that transform the application to be in this state. The subsequent sections of [Chapter 3](#) discuss each type of extraction in turn:

- Pulling up a UI component
- Pushing down a storage component
- Renaming a component

## 4.4 Refactoring: Pulling Up a UI Component —**TeamsAdmin**, **GamesAdmin**, **PredictionUI**, **WelcomeUI**

UI components are engines that only control controllers, views, and assets. Of the three, views and assets are optional. One special case of a UI component that does not contain views and assets is the API component, which serves up only API endpoints. We will cover API components in a future chapter.

Assets may be missing from a UI component when the component is part of an application where the various UI components use identical styles and JavaScript code. This is going to be the case for the UI components we are extracting in this chapter, as `WelcomeUI`, `TeamsAdmin`, `GamesAdmin`, and `PredictionUI` all share the same assets.

In this chapter, we are going to discuss only `GamesAdmin`. The extraction of the other UI components is analogous and their discussion would not provide much additional value. The resulting code from all the refactorings is available in the source code to this chapter.

The process for the extraction of a UI component follows the following steps, in which we

- *Generate* a new component
- *Move* files into the new component that make it its part of the application

- And fix tests by renaming references to component name and by moving, copying, and creating needed files

Finally, we'll discuss explicit contracts between components and the container app using the global navigation as an example.

#### 4.4.1 Generate

The first command to generate the new component should be familiar from [Chapter 2](#):

**Generate GamesAdmin component. Execute in ./**

[Click here to view code image](#)

```
$ rails plugin new components/games_admin --full \
  --mountable --skip-bundle --skip-git --skip-test \
  --skip-system-test --dummy-path=spec/dummy
```

We are creating a full and mountable engine. We are preventing the creation of a `.git` folder and of test classes.

Now, in order to be able to move all GamesAdmin-related files from AppComponent into the new component, we have to make sure that a number of folders exist in the new component's folder structure.

**Ensure needed directories exist. Execute in ./**

[Click here to view code image](#)

```
$ mkdir -p components/games_admin/app/views/games_admin
$ mkdir -p components/games_admin/spec/controllers/games_admin
$ mkdir -p components/games_admin/spec/features
```

#### 4.4.2 Move

After this, we are ready to move the bulk of the files that will make up the part of the application that is becoming GamesAdmin. In our case, everything that has to do with games' controllers and views must move into the new component. In terms of production code, that means we are moving `games_controller.rb` and the `games` folder under `/app/views`.

As for tests, we move controller, view, routing, and feature specs related to the GamesAdmin.

#### What If You Forget to Move a Class or Its Tests?

The penalty of forgetting a spec in the original engine is low: As we are extracting a component *upward*, the original component no longer has access to any of the extracted files and any remaining tests will fail in a very obvious way.

On the contrary, if we forget to pull a class up into the new component but otherwise refactor correctly, it might take some time before we notice. This is because the extracted component, by depending on the original component, has access to all the original files.

As we will see later, this behavior inverts when pushing a component down.

---

## Moving GamesAdmin files. Execute in ./

[Click here to view code image](#)

```
$ mv components/app_component/app/controllers/app_component/\
games_controller.rb \
  components/games_admin/app/controllers/games_admin/
$ mv components/app_component/app/views/app_component/games \
  components/games_admin/app/views/games_admin/games

$ mv components/app_component/spec/controllers/app_component/\
games_controller_spec.rb \
  components/games_admin/spec/controllers/games_admin/
$ mv components/app_component/spec/features/games_spec.rb \
  components/games_admin/spec/features
```

References to AppComponent are present throughout the moved files, most of which we can remove by using a combination of grep and sed. grep finds the files that need references replaced, and sed takes care of the actual rename. The naming rule discussed in [Section 2.1](#)—that is, don’t name your components badly—is in large part intended to support a large-scale refactoring like this. Naming that is too generic and causes homonyms makes this process much harder, as it prevents automatic renaming.

## Replacing all variations of AppComponent with analogous terms for GamesAdmin. Execute in ./

[Click here to view code image](#)

```
1 $ sed -i -- 's/ Game\./ AppComponent::Game\./g' components/\
2 games_admin/\
3 app/controllers/games_admin/games_controller.rb
4
5 $ grep -rl "module AppComponent" components/games_admin/ | \
6   xargs sed -i '' 's/module AppComponent/module GamesAdmin/g'
7 $ grep -rl "AppComponent::GamesController" components/games_admin/\
8   | xargs sed -i '' \
9     's/AppComponent::GamesController/GamesAdmin::GamesController/g'
10 $ grep -rl "AppComponent::Engine" components/games_admin/ | \
11   xargs sed -i '' 's/AppComponent::Engine/GamesAdmin::Engine/g'
12 $ grep -rl "app_component/" components/games_admin/ | \
13   xargs sed -i '' 's;app_component;games_admin;g'
14 $ grep -rl "app_component\." components/games_admin/ | \
15   xargs sed -i '' 's;app_component\.;games_admin\.;g'
```

After the scripted portion of the refactor, it is now time to fix the remaining parts that are currently preventing the component from working.

First, we fix the gemspec and Gemfile by adding needed external dependencies, CBRA components, and removing TODOs. Since we are extracting upward, GamesAdmin will depend on AppComponent. However, GamesAdmin does not directly depend on components AppComponent depends on (i.e., Predictor). Consequently, we add a dependency to app\_component to both files, but remove the

dependencies on Predictor.

As GamesAdmin is a UI component, it continues to depend on rails, slim-rails, and jquery-rails. Similarly, all the development dependencies stay the same.

## ./components/games\_admin/games\_admin.gemspec (partial) — dependencies section

[Click here to view code image](#)

```
1 s.add_dependency "rails", "5.1.4"
2 s.add_dependency "slim-rails", "3.1.3"
3 s.add_dependency "jquery-rails", "4.3.1"
4 s.add_dependency "app_component"
5
6 s.add_development_dependency "sqlite3"
7 s.add_development_dependency "rspec-rails"
8 s.add_development_dependency "shoulda-matchers"
9 s.add_development_dependency "database_cleaner"
10 s.add_development_dependency "capybara"
11 s.add_development_dependency "rails-controller-testing"
```

## ./components/games\_admin/Gemfile

[Click here to view code image](#)

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 path ".." do
6   gem "app_component"
7 end
8
9 gem "trueskill",
10   git: "https://github.com/benjaminleesmith/trueskill",
11   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

It should become second nature when extracting engine components to check that engine.rb under lib is looking good. We will almost always want to ensure that migrations are appended to whichever application loads the engine. Here, we are also setting up the generators to work the same as they work in AppComponent.

## ./components/games\_admin/lib/games\_admin/engine.rb

[Click here to view code image](#)

```
1 module GamesAdmin
2   class Engine < ::Rails::Engine
3     isolate_namespace GamesAdmin
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11  end
12
13  config.generators do |g|
```

```

13     g.orm           :active_record
14     g.template_engine :slim
15     g.test_framework  :rspec
16   end
17 end
18 end

```

Before we once again start running the tests to see how we are doing on the path to a working component, we create the spec helper. Since all AppComponent's tests passed before the refactoring, it makes sense to use its `spec_helper.rb` as a blueprint for the spec helper that we use for GamesAdmin as well.

`./components/games_admin/spec/spec_helper.rb`

[Click here to view code image](#)

```

1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10
11 require "rails-controller-testing"
12 Rails::Controller::Testing.install
13
14 Dir[GamesAdmin::Engine.root.join("spec/support/**/*.rb")].
15   each { |f| require f }
16
17 RSpec.configure do |config|
18   config.expect_with :rspec do |expectations|
19     expectations.
20       include_chain_clauses_in_custom_matcher_descriptions = true
21   end
22   config.mock_with :rspec do |mocks|
23     mocks.verify_partial_doubles = true
24   end
25
26   config.infer_spec_type_from_file_location!
27   config.disable_monkey_patching!
28   config.warnings = false
29   config.profile_examples = nil
30   config.order = :random
31   Kernel.srand config.seed
32
33   config.before(:suite) do
34     DatabaseCleaner.strategy = :transaction
35     DatabaseCleaner.clean_with(:truncation)
36   end
37
38   config.around(:each) do |example|
39     DatabaseCleaner.cleaning do
40       example.run
41     end

```

```

42   end
43
44   [:controller, :view, :request].each do |type|
45     config.include ::Rails::Controller::Testing::
46       TestProcess, :type => type
47     config.include ::Rails::Controller::Testing::
48       TemplateAssertions, :type => type
49     config.include ::Rails::Controller::Testing::
50       Integration, :type => type
51   end
52
53   config.include ObjectCreationMethods
54 end
55
56 Shoulda::Matchers.configure do |config|
57   config.integrate do |with|
58     with.test_framework :rspec
59     with.library :rails
60   end
61 end

```

This `spec_helper.rb` only differs from its original in line 14, where it references `GamesAdmin::Engine` instead of `AppComponent::Engine`.

After we create a `test.sh` file after the `AppComponent` version, we are ready to run the tests.

**`./components/games_admin/test.sh`**

[Click here to view code image](#)

```

1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running games admin engine specs"
6 bundle install | grep Installing
7 bundle exec rake db:create db:migrate
8 RAILS_ENV=test bundle exec rake db:create
9 RAILS_ENV=test bundle exec rake db:migrate
10 bundle exec rspec spec
11 exit_code+=$?
12
13 exit $exit_code

```

Time to start running the tests.

#### 4.4.3 Fix, Part 1: Reusing Test Helpers

The first run of `test.sh` in the component's folder brings to the surface a problem we have not seen before:

**Error when running `test.sh` for the first time (some results omitted). Execute in `./components/games_admin`**

[Click here to view code image](#)

\$ test.sh

```
...  
sportsball/components/games_admin/spec/spec_helper.rb:39:in  
`block in <top (required)>': uninitialized constant  
  ObjectCreationMethods (NameError)
```

The constant `ObjectCreationMethods` is referenced in the spec helper we just copied over. The test object creation methods are defined in `AppComponent` and not in `GamesAdmin`. The question now is: Do we remove that line from the spec helper in `GamesAdmin`, or not? Do we need to create these objects in our tests? The answer is, of course, yes. `games_controller_spec.rb` creates both teams and games to test the functionality of the extracted controller. Because of this, `GamesAdmin` tests need access to `AppComponent` test helpers.

---

## Exposing Multiple Requirable Parts of a Gem

When Ruby loads a gem, it adds its `lib` folder to a global variable `$LOAD_PATH`. This variable contains all the folders from which the current process loads files. The reason that we can write `require "rack"` after adding the `rack` gem to a `Gemfile` or `gemspec` is that `rack`'s `lib` folder contains a `rack.rb` file.

Any file within a gem's `lib` folder can be required in that way. For this reason, the conventional method is to only add one Ruby file, named the same as the gem, directly into the `lib` folder (<http://guides.rubygems.org/patterns/#loading-code>). Furthermore, to expose other requirable files, it is considered conventional to put these into `lib/GEMNAME`. That way these additional files can be required as `require GEMNAME/filename` (e.g., Rack utilities can be directly required with `require "rack/utils"`).

---

As the object creation methods are part of the test code of `AppComponent`, they are not automatically exposed when the gem is required. To expose them, we introduce a secondary requirable part of the `AppComponent` gem. To this end, we create a new file `test_helpers.rb` inside of `AppComponent`:

```
./components/app_component/lib/app_component/test_helpers.rb
```

[Click here to view code image](#)

```
1 require_relative "../../spec/support/object_creation_methods.rb"
```

This one line requires and thus ensures loading of `components/app_component/spec/support/object_creation_methods.rb`. This makes the `ObjectCreationMethods` module defined therein accessible to any gem or app that requires this file.

It is a good idea to namespace the constant `ObjectCreationMethods` under `AppComponent`. This prevents any naming collisions that might occur due to other components also defining object creation methods. A change of line 1 of `components/app_component/spec/`

support/object\_creation\_methods.rb to module AppComponent::ObjectCreationMethods takes care of this and allows the following, new version of GamesAdmin's spec helper.

**./components/games\_admin/spec/spec\_helper.rb**

[Click here to view code image](#)

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10
11 require "rails-controller-testing"
12 Rails::Controller::Testing.install
13
14 Dir[GamesAdmin::Engine.root.join("spec/support/**/*.rb")].
15   each { |f| require f }
16
17 require "app_component/test_helpers"
18
19 RSpec.configure do |config|
20   config.expect_with :rspec do |expectations|
21     expectations.
22       include_chain_clauses_in_custom_matcher_descriptions = true
23   end
24   config.mock_with :rspec do |mocks|
25     mocks.verify_partial_doubles = true
26   end
27
28   config.infer_spec_type_from_file_location!
29   config.disable_monkey_patching!
30   config.warnings = false
31   config.profile_examples = nil
32   config.order = :random
33   Kernel.srand config.seed
34
35   config.before(:suite) do
36     DatabaseCleaner.strategy = :transaction
37     DatabaseCleaner.clean_with(:truncation)
38   end
39
40   config.around(:each) do |example|
41     DatabaseCleaner.cleaning do
42       example.run
43     end
44   end
45
46   [:controller, :view, :request].each do |type|
47     config.include ::Rails::Controller::Testing::
48       TestProcess, :type => type
```

```

49 config.include ::Rails::Controller::Testing::
50   :TemplateAssertions, :type => type
51 config.include ::Rails::Controller::Testing::
52   :Integration, :type => type
53 end
54
55 config.include AppComponent::ObjectCreationMethods
56 end
57
58 Shoulda::Matchers.configure do |config|
59   config.integrate do |with|
60     with.test_framework :rspec
61     with.library :rails
62   end
63 end

```

Now, line 17 requires the test helpers and lines 51–52 ensures they are included in every test.

As we are doing this, we should not forget that the `spec_helper` inside of `AppComponent` needs to change in the same way. The new line for the inclusion of the object creation methods is `config.include AppComponent::ObjectCreationMethods` everywhere.

#### 4.4.4 Fix, Part 2: Routing Issues

The next run of `test.sh` sees every single test fail with some sort of routing problem.

The vast majority of errors stems from the new component not having any routes defined whatsoever. The route definition for `games` needs to be moved from `AppComponent` into `GamesAdmin` to make it look as follows (don't forget to remove the `games` resources from the `AppComponent` routes):

**`./components/games_admin/config/routes.rb`**

[Click here to view code image](#)

```

1 GamesAdmin::Engine.routes.draw do
2   resources :games
3 end

```

The next run of `test.sh` will show that at this point we have fixed all the specs of `GamesAdmin`. That means that, just like in the `Predictor` extraction, it is now time for going up the dependency chain to fix anything that requires the new engine. Here, that means that the container application is next.

#### 4.4.5 Fix, Part 3: The Container Application

There are a few changes that we already know need to happen from our previous work of extracting a component from an application: The `GamesAdmin` gem needs to be required and the engine needs to be mounted (see [Section 2.1](#)).

**`./Gemfile (partial)—just CBRA dependencies`**

[Click here to view code image](#)

```
1 path "components" do
2   gem "app_component"
3   gem "games_admin"
4 end
```

**./config/routes.rb**

[Click here to view code image](#)

```
1 Rails.application.routes.draw do
2   mount AppComponent::Engine, at: "/app_component"
3   mount GamesAdmin::Engine, at: "/games_admin"
4
5   root to: "app_component/welcome#show"
6 end
```

Now we can start running `test.sh` in the container application.

**Result of running `test.sh` in container app for the first time. Execute in `.`**

[Click here to view code image](#)

```
$ test.sh
Randomized with seed 34458
FFFF
```

Failures:

- 1) Engine 'App' has teams  
Failure/Error: visit "/"  
ActionView::Template::Error:  
 undefined local variable or method `games\_path' for  
 #<#<Class:0x007fcd19376188>:0x007fcd19374518>
- 2) Engine 'App' has games  
Failure/Error: visit "/"  
ActionView::Template::Error:  
 undefined local variable or method `games\_path' for  
 #<#<Class:0x007fcd19376188>:0x007fcd1e0c29a0>
- 3) Engine 'App' can predict  
Failure/Error: visit "/"  
ActionView::Template::Error:  
 undefined local variable or method `games\_path' for  
 #<#<Class:0x007fcd19376188>:0x007fcd19e9b590>
- 4) Engine 'App' hooks up to /  
Failure/Error: visit "/"  
ActionView::Template::Error:  
 undefined local variable or method `games\_path' for  
 #<#<Class:0x007fcd19376188>:0x007fcd19df89d0>

Something is not happy about the use of `games_path`. A quick search with `ack` yields a couple of locations where `games_path` is used.

**Searching for `games_path`. Execute in `.`**

[Click here to view code image](#)

```
$ ack games_path components/app_component/app/views/layouts/\
app_component/application.html.slim
32: li =link_to "Games", games_path
```

```

components/games_admin/app/views/games_admin/games/edit.html.slim
7:= link_to 'Back', games_path

components/games_admin/app/views/games_admin/games/new.html.slim
5:= link_to 'Back', games_path

components/games_admin/app/views/games_admin/games/show.html.slim
28:= link_to "Back", games_path, class: "button"

components/games_admin/lib/games_admin.rb
11:   {name: "Games", link: -> {::GamesAdmin::Engine.\
routes.url_helpers.games_path} }


```

Which file is the culprit? The only use outside of `games_admin` (where `games_path` is defined, because of the `games` resources' routes having been moved there earlier) is its occurrence in `application.html.slim`, which still resides in `AppComponent`. Opening the file quickly reveals these couple of lines as the cause of our test failures:

**`./components/app_component/app/views/layouts/app_component/application.html.slim` (partial)—the app's navigation bar**

[Click here to view code image](#)

```

1  section.top-bar-section
2    ul.left
3      li =link_to "Teams", teams_path
4      li =link_to "Games", games_path
5      li =link_to "Predictions", new_prediction_path

```

This is the code that defines that top navigation bar. On second thought, it is not surprising that when teasing apart the front end, we are running into trouble where the front end is bound together: in its navigational infrastructure.

What is happening here? This piece of code from `AppComponent` is dependent on a path that is now being defined in `GamesAdmin`. As `GamesAdmin` depends on `AppComponent`, `AppComponent` *cannot* depend on `GamesAdmin` to get access to this path. It is a rather nasty case of a seemingly necessary circular dependency.

The resolution of this problem will require a deeper discussion and more analysis in one of the subsequent chapters of this book. Here, to be able to finish the refactoring, we can simply slash through the Gordian knot: Replace the usage of `games_path` with the path of the URL that it resolves to.

Originally, `games_path` pointed to `/games`, which pointed to the `index` action of `GamesController`. Since that controller moved into `GamesAdmin`, its new path is `/games_admin/games`. If we replace the line that defines the link to the `games` section of the app, we can hope to solve the problem with the tests.

## AppComponent navigation bar with static games path

[Click here to view code image](#)

```

1  section.top-bar-section
2    ul.left

```

```
3   li =link_to "Teams", teams_path
4   li =link_to "Games", "/games_admin/games"
5   li =link_to "Predictions", new_prediction_path
```

Another run of `test.sh` reveals that we have almost, but not quite, fixed the tests.

## Result of running `test.sh` in container app for the second time. Execute in `./`

[Click here to view code image](#)

```
$ test.sh
Randomized with seed 33266
.F..
```

Failures:

- 1) Engine 'App' has games  
Failure/Error: within "main h1" do  
Capybara::ElementNotFound:  
 Unable to find css "main h1"

Notice that this is the feature spec that verifies that the games section of the application works. But why is it broken? Has our layout changed? Why would `main h1` suddenly no longer be on the page? Best to not guess; instead, we should check by verifying what is going in the browser ourselves—let's run `rails s`.

[Figure 4.12](#) is what we see when we click on “Games” in the navigation bar.



**Figure 4.12.** The games admin looking pretty frugal

Obviously, games admin looks pretty awful. It has lost the previous somewhat more colorful layout and seems to instead use the default Rails layout. That makes sense when you think back to how engines are created (see Appendix A): The generation of an engine also creates a layout and the engine uses it by default.

In other applications, where the supported use case diverges more, different sections of the app might allow (or require) for separate layouts. However, in our case, we want GamesAdmin to use the same layout as AppComponent (for now, at any rate). So, let's delete the generated layout:

## Deleting the layout in GamesAdmin

[Click here to view code image](#)

```
rm -rf components/games_admin/app/views/layouts
```

Deleting the layout doesn't yet make GamesAdmin use the layout defined in

AppComponent. For that, we need to invoke the layout method—a class method on ActionController::Base from which all our controllers inherit. Using layout, we can specify the path of the layout we want to use for a controller:

## Setting the AppComponent layout for GamesAdmin controllers

[Click here to view code image](#)

```
1 module GamesAdmin
2   class ApplicationController < ActionController::Base
3     layout "app_component/application"
4   end
5 end
```

Let's once again try running test.sh:

## Result of running test.sh in container app for the third time. Execute in . /

[Click here to view code image](#)

```
$ test.sh
Randomized with seed 42316
.F..
```

Failures:

```
1) Engine 'App' has games
Failure/Error: click_link "Games"
ActionView::Template::Error:
  undefined local variable or method `teams_path' for
  #<#<Class:0x007fc0bba84660>:0x007fc0bba7f4d0>
```

So, now teams\_path is undefined? How do we keep running into these path issues around GamesAdmin? The error we are seeing is very much like the one before where games\_path was undefined. If we analyze what we just did again, we realize that we are now using the same layout in two different engines: GamesAdmin and AppComponent. The routes defined in these two engines are totally disjunct: games is defined in the former, teams and predictions in the latter.

The test failure we are seeing is a test that clicks on “Games” in the navigation bar and thus navigates into a section of the app that is provided by GamesAdmin. And within GamesAdmin, teams\_path is, as we just saw, not defined.

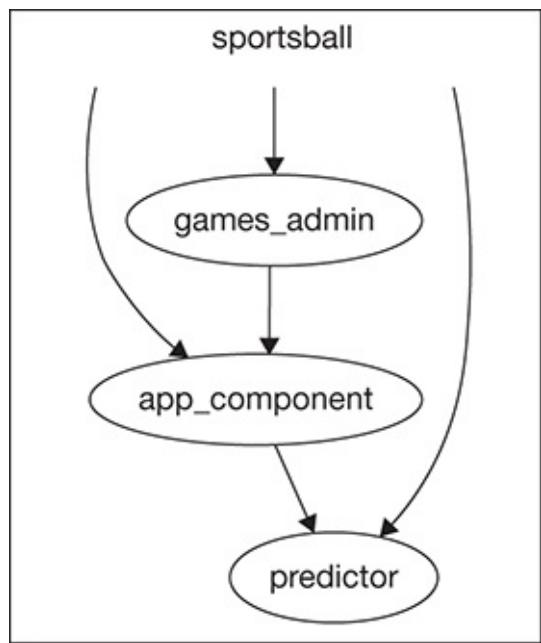
We can conclude from this that our previous fix to the navigation bar did not go far enough. We need all the paths in the navigation bar to be hard-coded until we solve this entangled circular dependency problem.

## ./components/app\_component/app/views/layouts/app\_component/app\_component.html.erb - the new AppComponent navigation bar

[Click here to view code image](#)

```
1   section.top-bar-section
2     ul.left
3       li =link_to "Teams", "/app_component/teams"
4       li =link_to "Games", "/games_admin/games"
5       li =link_to "Predictions",
```

Now, finally, `test.sh` shows that our container application passes all its tests. It is prudent to check that everything is fine using `./build.sh`, which in fact it is. The dependency graph of this stage of the extraction process is depicted in [Figure 4.13](#).



**Figure 4.13.** Sportsball dependencies with GamesAdmin extracted

#### 4.4.6 An Explicitly Contracted Global Navigation

To remove the circular dependency created through the global navigation, we can employ some duck typing for our UI components. The basic idea is that instead of `AppComponent`'s containing explicit knowledge about the UI components, it should be given a configuration for those components from which it can build the navigation. In strongly typed languages, one might create a `NavigationBarInterface` as well as a `NavigationBarEntryInterface`. Since there are no interfaces in Ruby, let's start with a test that verifies the existence and structure of a navigation bar entry for `GamesAdmin`.

`./components/games_admin/spec/nav_entry_spec.rb`

[Click here to view code image](#)

```
1 RSpec.describe "nav entry" do
2   it "points at the list of games" do
3     entry = GamesAdmin.nav_entry
4     expect(entry[:name]).to eq "Games"
5     expect(entry[:link].call).to eq "/games_admin/games"
6   end
7 end
```

The way this test is written, it requires us to add a method to the `GamesAdmin` module, like so:

`./components/games_admin/lib/games_admin.rb`

[Click here to view code image](#)

```
1 require "slim-rails"
2 require "jquery-rails"
3
4 require "app_component"
5
6 module GamesAdmin
7   require "games_admin/engine"
8
9   def self.nav_entry
10    {
11      name: "Games",
12      link: -> {::GamesAdmin::Engine.routes.url_helpers.games_path}
13    }
14  end
15 end
```

If we assume this is also done for the other components represented in the global nav, namely `TeamsAdmin` and `PredictionUI`, we can create a global navigation configuration in the container app.

`./config/initializers/global_navigation.rb`

[Click here to view code image](#)

```
1 Rails.application.config.main_nav =
2   [
```

```

3     TeamsAdmin.nav_entry,
4     GamesAdmin.nav_entry,
5     PredictionUi.nav_entry
6   ]

```

With this config in place, AppComponent, which holds the application layout for all these components, can construct the navigation bar without having to know about any of the UI components directly.

`./components/app_component/app/views/layouts/app_component/application.html.slim`

[Click here to view code image](#)

```

1 - Rails.application.config.main_nav.each do |nav_entry|
2   li =link_to nav_entry[:name], nav_entry[:link].call

```

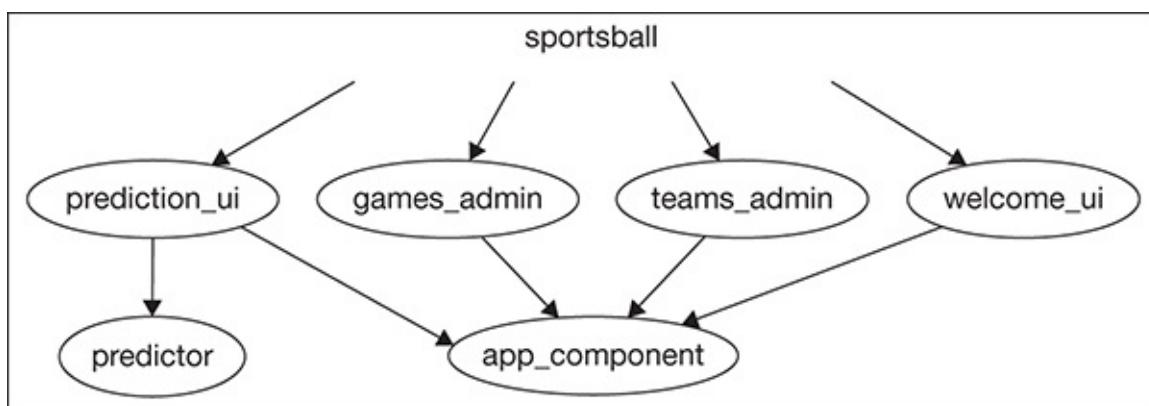
Note that the container app now must know about the structure of the global navigation object and the location of the components' navigation configurations, but not about the form of the individual navigation entries. GamesAdmin and the other UI components know about the structure of their individual navigation entries only. Finally, AppComponent must know both the structure of the navigation as well as its entries. Both the container app and AppComponent know about where to find the global navigation component. This last coupling could be removed from AppComponent by “dependency-injecting” the config from the container application into AppComponent.

## All UI Components Extracted

There are two directories for this chapter in the source code repository (<https://github.com/shageman/component-based-rails-applications-book>).

c4s04\_book contains only the refactoring discussed in this chapter. c4s04\_finished contains the codebase with all UI components extracted. It is the latter version that will be the basis for subsequent sections.

With all UI components extracted, [Figure 4.14](#) depicts the dependency graph of the Sportsball application. Note that with these changes the container app no longer needs to depend on AppComponent directly. Also note, that AppComponent is no longer depending on Predictor, but rather PredictionUI is.



**Figure 4.14.** Sportsball dependencies with extracted UI components

## 4.5 Refactoring: Pushing Down a Model Component —Teams, Games

In a wide sense, model components contain classes that store application information. In a narrower sense, model engines, as we are going to discuss in this chapter, are engines that contain ActiveRecord models.

As we discussed in the previous chapters, the models Team and Game are needed by the views and controllers of the application, but the models themselves don't need any of those views or controllers in return. Hence, the direction for the extraction of the model layer in Sportsball is *down*. As such, the refactoring motion is the opposite to that of the UI component.

By extracting the two ActiveRecord models into separate components, we are creating structure in the model layer. This is the first answer componentization has for the common criticism of overly complex model structures in large Rails code bases. More can be done, but that will have to be a topic for future chapters.

*Generate, move, and fix* are once again the steps to implementing the new component. The first steps of the process are analogous to the UI component extraction from the last chapter: Generate the engine, create directories, move files, create test script, and add dependencies to `gemspec` and `Gemfile`.

### Generate the teams engine. Execute in ./

[Click here to view code image](#)

```
$ rails plugin new components/teams \
  --skip-yarn --skip-git --skip-action-mailer --skip-action-cable \
  --skip-puma --skip-sprockets --skip-spring --skip-listen \
  --skip-coffee --skip-javascript --skip-turbolinks \
  --skip-test --skip-system-test \
  --dummy-path=spec/dummy --full --mountable
```

### Ensure needed directories exist. Execute in ./

[Click here to view code image](#)

```
$ mkdir -p components/teams/app/models/teams
$ mkdir -p components/teams/spec/models/teams
$ mkdir -p components/teams/spec/support
```

### Move Team, its spec, and rename their module to Teams. Execute in ./

[Click here to view code image](#)

```
$ mv components/app_component/app/models/app_component/team.rb \
  components/teams/app/models/teams/
```

```
$ mv components/app_component/spec/models/app_component/team_spec.rb \
  components/teams/spec/models/teams/
```

```
$ grep -rl "AppComponent" components/teams/ | \
  xargs sed -i '' 's/module AppComponent/module Teams/g'
```

## Create ./components/teams/test.sh

[Click here to view code image](#)

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running teams engine specs"
6 bundle install | grep Installing
7 bundle exec rake db:create db:migrate
8 RAILS_ENV=test bundle exec rake db:create
9 RAILS_ENV=test bundle exec rake db:migrate
10 bundle exec rspec spec
11 exit_code+=$?
12
13 exit $exit_code
```

For tests to run, we once again need to create `spec_helper.rb` first. We can use `AppComponent`'s `spec_helper` as a template and realize that we will need to pull the team-related object creation methods into this component, too. Make a mental note; we'll get back to that.

## ./components/teams/spec/spec\_helper.rb

[Click here to view code image](#)

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "ostruct"
9
10 Dir[Teams::Engine.root.join("spec/support/**/*.rb")].
11   each { |f| require f }
12
13 RSpec.configure do |config|
14   config.expect_with :rspec do |expectations|
15     expectations.
16       include_chain_clauses_in_custom_matcher_descriptions = true
17   end
18   config.mock_with :rspec do |mocks|
19     mocks.verify_partial_doubles = true
20   end
21
22   config.infer_spec_type_from_file_location!
23   config.disable_monkey_patching!
24   config.warnings = false
25   config.profile_examples = nil
26   config.order = :random
27   Kernel.srand config.seed
28
29   config.before(:suite) do
30     DatabaseCleaner.strategy = :transaction
31     DatabaseCleaner.clean_with(:truncation)
```

```

32   end
33
34   config.around(:each) do |example|
35     DatabaseCleaner.cleaning do
36       example.run
37     end
38   end
39
40   config.include Teams::ObjectCreationMethods
41 end
42
43 Shoulda::Matchers.configure do |config|
44   config.integrate do |with|
45     with.test_framework :rspec
46     with.library :rails
47   end
48 end

```

### 4.5.1 Getting Away with Less

The next opportunity for an interesting decision comes when we have to fix gemspec and Gemfile. So far, every component that was an engine required all of Rails. If this component only cares about ActiveRecord, do we need to include all of Rails? Or can we get away with less? Let's find out and only include `active_record` in both gemspec and Gemfile.

`./components/teams/teams.gemspec`

[Click here to view code image](#)

```

1 $:.push File.expand_path("../lib", __FILE__)
2
3 # Maintain your gems version:
4 require "teams/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name          = "teams"
9   s.version       = Teams::VERSION
10  s.authors      = ["Stephan Hagemann"]
11  s.email         = ["stephan.hagemann@gmail.com"]
12  s.homepage     = ""
13  s.summary       = "Summary of Teams."
14  s.description   = "Description of Teams."
15
16  s.files = Dir["{app,config,db,lib}/**/*",
17    "MIT-LICENSE", "Rakefile", "README.md"]
18
19  s.add_dependency "activerecord", "5.1.0"
20
21  s.add_development_dependency "sqlite3"
22  s.add_development_dependency "rspec-rails"
23  s.add_development_dependency "shoulda-matchers"
24  s.add_development_dependency "database_cleaner"
25 end

```

## **./components/teams/Gemfile**

[Click here to view code image](#)

```
1 source "https://rubygems.org"
2
3 gemspec
```

If, at this point, we run the specs, we get a bunch of errors related to active\_job/railtie:

## **ActionMailer railties are missing. Execute in ./components/teams**

[Click here to view code image](#)

```
$ test.sh
*** Running teams engine specs
rake aborted!
LoadError: cannot load such file -- active_job/railtie
  from ./components/teams/spec/dummy/config/application.rb:9
```

The standard spec dummy loads all of Rails by default, as can be seen in ./spec/dummy/config/application.rb. In our current case, it loads a little less, because action\_view, sprockets, and test\_unit were already excluded when the engine was generated. We will have to comment out all other the parts that are no longer available to this dummy app, because we didn't include all of Rails: action\_mailer and action\_pack (the latter is in fact available to the dummy app in test mode as it is a dependency of rspec-rails, but we should comment it out anyway—we don't expect to need it). This leaves us with the following application.rb:

## **./components/teams/spec/dummy/config/application.rb**

[Click here to view code image](#)

```
1 require_relative "boot"
2
3 # Pick the frameworks you want:
4 require "active_record/railtie"
5 # require "action_controller/railtie"
6 # require "action_view/railtie"
7 # require "action_mailer/railtie"
8 # require "active_job/railtie"
9 # require "action_cable/engine"
10 # require "rails/test_unit/railtie"
11 # require "sprockets/railtie"
12
13 Bundler.require(*Rails.groups)
14 require "teams"
15
16 module Dummy
17   class Application < Rails::Application
18     # Initialize configuration defaults for
19     # originally generated Rails version.
20     config.load_defaults 5.1
21
```

```

22  # Settings in config/environments/* take precedence over
23  # those specified here.
24  # Application configuration should go into files in
25  # config/initializers -- all .rb files in that directory
26  # are automatically loaded.
27 end
28 end

```

Depending on the exact version of Rails you are using, you may need to remove references to configurations for—now unavailable and unknown—Rails gems from some or all of the config files in ./components/teams/spec/dummy/config/environments/.

#### 4.5.2 Fixing Teams Tests

A quick check of the current state of the test reveals that the object creation methods can't be found.

**Run tests: Team's object creation methods missing (some results omitted). Execute in ./components/teams**

[Click here to view code image](#)

```

$ test.sh
*** Running teams engine specs

. .
./sportsball/components/teams/spec/spec_helper.rb:37:
in `block in <top (required)>':
uninitialized constant Teams::ObjectCreationMethods (NameError)

```

The object creation methods for Team need to move from AppComponent into the Teams engine, and, analogous to the previous section, be exposed through Teams engine test helpers.

**./components/teams/spec/support/object\_creation\_methods.rb**

[Click here to view code image](#)

```

1 module Teams::ObjectCreationMethods
2   def new_team(overrides = {})
3     defaults = {
4       name: "Some name #{counter}"
5     }
6     Teams::Team.new { |team| apply(team, defaults, overrides) }
7   end
8
9   def create_team(overrides = {})
10    new_team(overrides).tap(&:save!)
11  end
12
13 private
14
15   def counter
16     @counter ||= 0
17     @counter += 1
18   end

```

```

19
20 def apply(object, defaults, overrides)
21   options = defaults.merge(overrides)
22   options.each do |method, value_or_proc|
23     object. __send__ (
24       "#{method}=",
25       value_or_proc.is_a?(Proc) ?
26         value_or_proc.call : value_or_proc)
27   end
28 end
29 end

```

**./components/teams/lib/teams/test\_helpers.rb**

[Click here to view code image](#)

```
1 require_relative "../../spec/support/object_creation_methods.rb"
```

**./components/app\_component/spec/support/object\_creation\_methods.rb**

[Click here to view code image](#)

```

1 module AppComponent::ObjectCreationMethods
2   def new_game(overrides = {})
3     defaults = {
4       first_team: -> { new_team },
5       second_team: -> { new_team },
6       winning_team: 2,
7       first_team_score: 2,
8       second_team_score: 3,
9       location: "Somewhere",
10      date: Date.today
11    }
12
13    AppComponent::Game.new do |game|
14      apply(game, defaults, overrides)
15    end
16  end
17
18  def create_game(overrides = {})
19    new_game(overrides).tap(&:save!)
20  end
21
22  private
23
24  def counter
25    @counter ||= 0
26    @counter += 1
27  end
28
29  def apply(object, defaults, overrides)
30    options = defaults.merge(overrides)
31    options.each do |method, value_or_proc|
32      object. __send__ (
33        "#{method}=",
34        value_or_proc.is_a?(Proc) ?
35          value_or_proc.call : value_or_proc)
36    end
37  end

```

38 `end`

At this point, running the tests reveals that not all occurrences of `AppComponent` have been removed from `Teams`.

## Run tests: Incorrect namespace. (some results omitted). Execute in `./components/teams`

[Click here to view code image](#)

```
$ test.sh
*** Running teams engine specs

. . .

./sportsball/components/teams/spec/models/teams/team_spec.rb:1:
  in `<top (required)>':
    uninitialized constant AppComponent (NameError)
```

Since all references to `AppComponent` inside of `Teams` are in relation to `Team`, we can use the following commands to correct all occurrences of `AppComponent::Team` to `Teams::Team`. Be sure to execute this in the root of the application to fix all references everywhere in the app.

## Rename `AppComponent::Team` to `Teams::Team`. Execute in `./`

[Click here to view code image](#)

```
$ grep -rl "AppComponent::Team" components/teams/ | \
  xargs sed -i '' 's/AppComponent::Team/Teams::Team/g'
```

Next, the tests indicate that we have yet to fix our schema to reflect the new fully qualified name of `Teams::Team` in the database.

## Run tests. Execute in `./components/teams`

[Click here to view code image](#)

```
$ test.sh
*** Running teams engine specs
```

Randomized with seed 20999

F

Failures:

```
1) Teams::Team
Failure/Error: it { should validate_presence_of(:name) }
ActiveRecord::StatementInvalid:
  Could not find table 'teams_teams'
```

We can leverage the power of migrations to make sure that database management continues to be straightforward and in sync with the CBRA approach:

1. Component refactorings (whether it be renaming or extracting) should not lead to inconsistencies between the names of models and the names of their tables.
2. A component should be in control of *its* migrations (e.g., any migrations related to `Team` should be in `Teams`).

### 3. Refactorings should not break past migrations' correctness.

Without point 1, the internal mismatch of model and table names would be increasing with growing componentization. That would lead to higher cognitive costs with every added component and every additional refactoring. Such a situation would be an unacceptable cost to the entire approach for long-term maintainability. Point 2 is related in that there would be growing cognitive overhead to the structure of models and their underlying table structure. Components wouldn't be self-contained in that they would have to rely on *something else* to define the correct schema for them (e.g., if migrations were left in the container app, components couldn't be run using their dummy app without hacking the DB back into place—this would also constitute a circular dependency between the container app and the component). Point 3 is necessary as without it the cost of additional migrations would be dependent on the number of previous migrations—once again, an unacceptable effect of componentization.

Both points 2 and 3 are maintained if we move the Team migration from AppComponent into Teams as follows.

#### **Moving existing Team migration into Teams. Execute in ./**

[Click here to view code image](#)

```
$ mkdir -p components/teams/db/migrate
$ mv components/app_component/db/migrate/\
*_create_app_component_teams.rb components/teams/db/migrate
```

Because migrations are timestamped, they remain in the previous order and execute correctly when moved into a different location. Of course, we must ensure that the migration is still found and that their content does not change.

At this point, Teams has a migration that will create a table called app\_component\_team. By adding a new migration that changes the table's name to teams\_team, point 1 is achieved.

```
./components/teams/db/migrate/GENERATED_DATE_move_team_from_a
component_to_teams.rb - new migration renaming app_component_team into
teams_team
```

[Click here to view code image](#)

```
1 class MoveTeamFromAppComponentToTeams < ActiveRecord::Migration[5.0]
2   def change
3     rename_table :app_component_teams, :teams_teams
4   end
5 end
```

#### **Teams' tests are fixed. Execute in ./components/teams**

[Click here to view code image](#)

```
$ test.sh
*** Running teams engine specs
== 20150110181102 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
  -> 0.0018s
== 20150110181102 CreateAppComponentTeams: migrated (0.0019s) =====
```

```

== 20150825215500 MoveTeamFromAppComponentToTeams: migrating =====
-- rename_table(:app_component_teams, :teams_teams)
-> 0.0007s
== 20150825215500 MoveTeamFromAppComponentToTeams: migrated (0.0007s)

== 20150110181102 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
-> 0.0011s

== 20150110181102 CreateAppComponentTeams: migrated (0.0012s) =====

== 20150825215500 MoveTeamFromAppComponentToTeams: migrating =====
-- rename_table(:app_component_teams, :teams_teams)
-> 0.0006s
== 20150825215500 MoveTeamFromAppComponentToTeams: migrated (0.0007s)

Randomized with seed 26777
.

Finished in 0.09969 seconds (files took 1.12 seconds to load)
1 example, 0 failures

```

Randomized with seed 26777

As before, we adapt our component's engine.rb to append the engine's migrations to any application loading this engine. This ensures that AppComponent and the container app will use the two migrations now managed within Teams.

**./components/teams/lib/teams/engine.rb**

[Click here to view code image](#)

```

1 module Teams
2   class Engine < ::Rails::Engine
3     isolate_namespace Teams
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11
12    config.generators do |g|
13      g.orm             :active_record
14      g.test_framework :rspec
15    end
16  end
17 end

```

### 4.5.3 Fixing Everything Else

The new component works in isolation, but likely not yet in conjunction with all the other components to construct the functionality of the Sportsball app. To make it work, we need to make the components that are dependent on Teams aware of the new gem and configure them to use it appropriately.

But which components must be adapted?

If we did everything right in the extraction so far, it should be the components that make direct use of the capabilities of stuff that are now in Teams. We could simply run the tests of every component and see if they are currently broken (e.g., the tests for AppComponent are currently broken with errors like the following):

**Broken tests of AppComponent (some results omitted). Execute in  
./components/app\_component**

[Click here to view code image](#)

```
$ test.sh

. . .

1) AppComponent::Game should belong to second_team
Failure/Error: it { should belong_to :second_team }
Expected AppComponent::Game to have a belongs_to \
association called second_team (Teams::Team does not exist)
```

GamesAdmin currently fails because of the following two problems.

**Broken tests of GamesAdmin (some results omitted). Execute in  
./components/games\_admin**

[Click here to view code image](#)

```
$ test.sh

. . .

1) games_admin/games/show renders attributes in <p>
Failure/Error: @game = assign(:game, create_game(\n
    location: "Location"))
NoMethodError:
  undefined method `new_team' for \
  #<RSpec::ExampleGroups::GamesAdminGamesShow:0x007fca9c5a13d8>

2) games admin allows for the management of games
Failure/Error: team1 = create_team name: "UofL"
NoMethodError:
  undefined method `create_team' for \
  #<RSpec::ExampleGroups::GamesAdmin:0x007fca9c573de8>
```

Think back to the component extraction and the refactorings of this chapter to realize that the existence of Teams::Team and its object creation methods new\_team and create\_team are the only outward-facing contributions of the Teams component.

Because of this, one could also find all the components directly using Teams via ack ( (new|create) \_team) and ack Teams::Team.

Either method reveals that a total of four engines currently need Teams directly:

1. AppComponent
2. GamesAdmin
3. PredictorUi
4. TeamsAdmin

For all these components, the steps to correctly load Teams are exactly the same. We will list them here for AppComponent. The completed fixes for all components can be found in the source code to this chapter.

1. List the teams gem in the gemspec.

**New line for ./components/app\_component/app\_component.gemspec**

[Click here to view code image](#)

```
1 s.add_dependency "teams"
```

2. Add the correct path for the teams into Gemfile.

**./components/app\_component/Gemfile**

[Click here to view code image](#)

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 path "../" do
6   gem "teams"
7 end
8
9 gem "trueskill",
10   git: "https://github.com/benjaminleesmith/trueskill",
11   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

3. Ensure the teams gem is required by the component.

**./components/app\_component/lib/app\_component.rb**

[Click here to view code image](#)

```
1 require "slim-rails"
2 require "jquery-rails"
3
4 require "teams"
5
6 module AppComponent
7   require "app_component/engine"
8 end
```

4. Add Teams' test helpers into the spec helper of the component.

**./components/app\_component/spec/spec\_helper.rb**

[Click here to view code image](#)

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", FILE)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10 require "ostruct"
11
12 require "rails-controller-testing"
13 Rails::Controller::Testing.install
14
15 Dir[AppComponent::Engine.root.join("spec/support/**/*.rb")].
16   each { |f| require f}
17
18 require "teams/test_helpers"
19
20 RSpec.configure do |config|
21   config.expect_with :rspec do |expectations|
22     expectations.
23       include_chain_clauses_in_custom_matcher_
24   descriptions = true
25 end
26 config.mock_with :rspec do |mocks|
27   mocks.verify_partial_doubles = true
28 end
29 config.infer_spec_type_from_file_location!
30 config.disable_monkey_patching!
31 config.warnings = false
32 config.profile_examples = nil
33 config.order = :random
34 Kernel.srand config.seed
35
36 config.before(:suite) do
37   DatabaseCleaner.strategy = :transaction
38   DatabaseCleaner.clean_with(:truncation)
39 end
40
41 config.around(:each) do |example|
42   DatabaseCleaner.cleaning do
43     example.run
44   end
45 end
46
47 [:controller, :view, :request].each do |type|
48   config.include ::Rails::Controller::Testing::
49     TestProcess, :type => type
50   config.include ::Rails::Controller::Testing::
51     TemplateAssertions, :type => type
52   config.include ::Rails::Controller::Testing::
53     Integration, :type => type
54 end
```

```

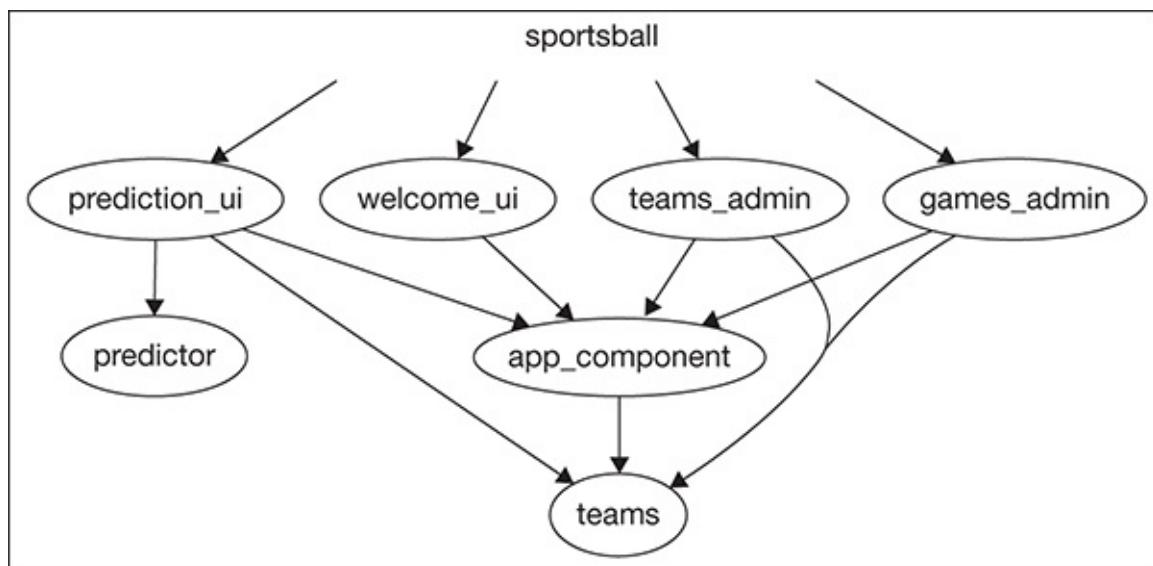
55
56 config.include Teams::ObjectCreationMethods
57 config.include AppComponent::ObjectCreationMethods
58 end
59
60 Shoulda::Matchers.configure do |config|
61   config.integrate do |with|
62     with.test_framework :rspec
63     with.library :rails
64   end
65 end

```

Repeat this for all four components and you are done. Run `./build.sh` to confirm.

Note that we did not add the `teams` gem to the `Gemfile` of the container app. Because we have been using the path block syntax consistently, the container app is looking for dependencies of its direct CBRA dependencies in the same folder and thus automatically finds `teams`—no need to add this transitive dependency.

With this refactoring completed, the dependency structure of Sportsball looks as depicted in [Figure 4.15](#).

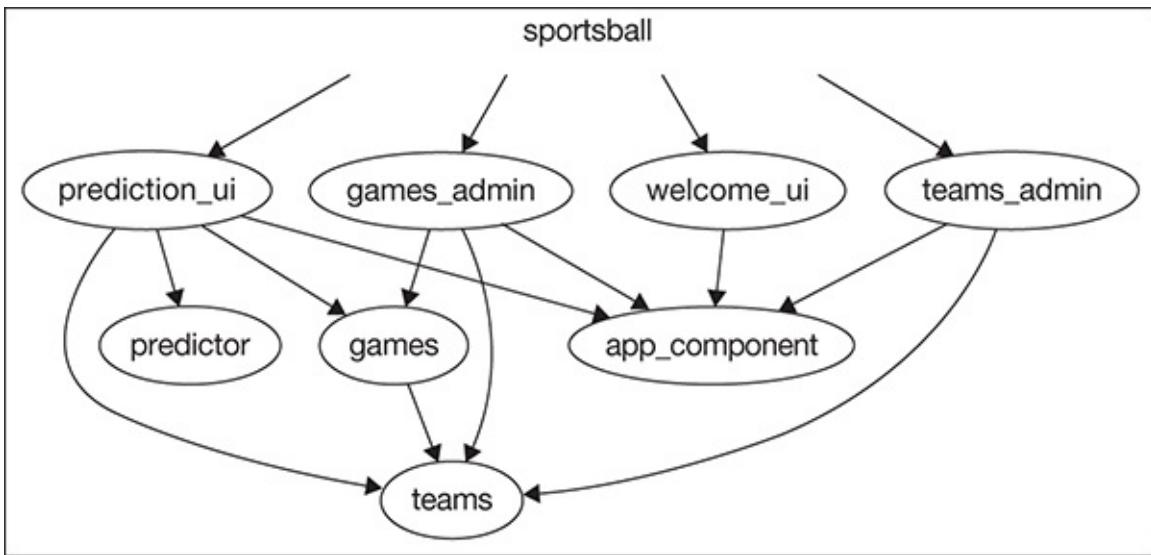


**Figure 4.15.** Sportsball dependencies with Teams components

## All Model Components Extracted

Analogous to [Section 4.4](#), there are two directories for this chapter in the source code repository (<https://github.com/shageman/component-based-rails-applications-book>). `c4s05_book` contains only the refactoring discussed in this chapter. `c4s05_finished` contains the codebase with all model components extracted. It is the latter version that will be the basis for subsequent sections.

With all model components extracted, [Figure 4.16](#) shows the dependency graph of the Sportsball application.



**Figure 4.16.** Sportsball dependencies with extracted model components

## 4.6 Refactoring: Improve Naming of Component —AppComponent to WebUI

We will finish out this series of refactorings with one that may seem too trivial for some to mention: renaming a component. Arguably, this is the most important refactoring of them all. It is the refactoring that aligns the name of the component with what it is doing. The function of a component may change due to an internal change to its features, because of functionality extraction, or because functionality was added. Ensuring that its name reflects *what it is about* is paramount to maintaining the benefits of a component-based architecture.

Then again, you never really questioned this, right? You always refactor all your classes to have the best name possible? You give methods and variables names that reflect their intent? Good.

We will find that refactoring of component names is not as well supported by IDEs as refactorings of class, method, and variable names. In fact, we already saw this when discussing the previous extraction refactorings. As every extraction renames the namespace of the extracted classes, they all come with a significant amount of work related to just the renaming of the component and its namespace.

### 4.6.1 Whatever Happened to AppComponent?

The previous three types of refactorings extracted a total of seven components, all from AppComponent. Over the course of these refactorings, AppComponent lost all of its models, controllers, helpers, migrations, and tests. It also lost all views except for the application layout. Only the assets were not touched by any of the refactorings.

AppComponent was never a good name, and it has not gotten better with any of the aforementioned refactoring steps. So, what does it do now? What should its name be?

Whereas at the beginning AppComponent did everything that the app did, it now has a

very narrow and focused contribution: It creates the base layout, style, and JavaScript framework for the application. Its name should be something that reflects that changed role. I like `WebUserInterface` or `WebUi` as a name. These names capture that everything this component is about is related to the user interface of the application. It does not quite capture that no specific user interfaces are in it (i.e., that everything contained is in fact the basis for the UI of other components). However, I personally do not think trying to capture that with something like `BaseWebUi` adds enough clarity that it merits the longer name.

Important for this decision is that the team can agree on a term and shares an understanding of how it is to be used within the application. Even more important is to not hold on to easily changeable decisions too strongly. And, as we will see in the following section, the name of a component is a fairly easily changeable decision.

#### 4.6.2 Refactoring Component Names

With every extraction we discussed in the previous sections, we moved classes from one component to a component with a different name. In that sense, we have already fully discussed the rename refactor.

To not repeat ourselves, let's discuss the high-level steps in a component renaming refactor, knowing that we have already seen every one of these steps in practice in a previous section.

First, the most basic description of the refactor is that we want to refactor from a working component A to a working component B. Since components are represented in gems and gems, by default, use their own name within their own code, we need to replace all mentions of A with the analogous mention of B. Specifically, that applies to the following possible occurrences:

- Component name in folder names
- Component name in file names
- Component name within files
- Component name in migrations

In the middle of the model component extraction we discussed in [Section 4.5](#), `AppComponent` had the following folder structure.

#### 4.6.2.1 Renaming Folders

##### Folder structure of **AppComponent** during extraction of model components

[Click here to view code image](#)

```
./components/app_component/
└── app
    ├── assets
    │   └── images
    │       └── app_component
    ├── javascripts
    │   └── app_component
    ├── stylesheets
    │   └── app_component
    ├── controllers
    │   └── app_component
    ├── models
    │   └── app_component
    └── views
        └── layouts
            └── app_component
└── bin
└── config
    └── initializers
└── lib
    └── app_component
        └── asks
└── spec
    ├── dummy
    └── models
        └── app_component
        └── support
└── vendor
└── assets
    ├── javascripts
    │   └── app_component
    └── stylesheets
        └── app_component
```

There are ten folders named `app_component`. Closer investigation reveals that the files in `controllers` and `models` are not actually used. These folders can safely be deleted. All the other folders need to be renamed to `web_ui` for the rename refactor.

#### 4.6.2.2 Renaming Files

At the same state of the refactor, there are four files that contain `app_component` in their name. All files need their name to change from `*app_component*` to the analogous name with `web_ui`.

##### File names in **AppComponent** containing `app_component` during extraction of model components. Execute in `./`

[Click here to view code image](#)

```
$ find . -name \*app_component\* -type f -print
```

```
./components/app_component/app/assets/config/\
    app_component_manifest.js
./components/app_component/app_component.gemspec
./components/app_component/lib/app_component.rb
./components/app_component/lib/tasks/app_component_tasks.rake
```

#### 4.6.2.3 Renaming Within Files

Within files, any mention of `AppComponent` needs to be replaced with `WebUi` and every mention of `app_component` with `web_ui`. There are quite a few files that can be skipped, however. For example, the following files can be skipped as they are cache and log files whose content is generated by the application:

- `*.log`
- `sprockets/**/*`
- `tmp/cache/**/*`

There are a few files that *need* to be skipped for the process to work:

- `*.sqlite` – binary files
- `assets/images/**/*` – binary files
- `schema.rb` – we do not want to change the schema this way (see the following)
- `structure.sql` – we do not want to change the schema this way (see the following)
- `db/migrate/*` – we do not want to change the schema this way (see the following)

It is this step of changing the component name within files that benefits the most from good naming. As we discussed before, if the component name is unique, this step can be performed without fear of the homonym problem, which would force us to decide for every occurrence of the term whether it is in fact the component name and whether it needs to be changed.

#### 4.6.2.4 Renaming Database Tables

The database schema and migrations need to be handled differently from files and folders if they contain the component name. The difference stems from the fact that, first, the schema is autogenerated (so we actually do not want to change anything in it), and second, that migrations are a historical list of all the state changes of the database schema. As such, we do not ever want to change existing migrations, but instead we want to add new migrations that create the intended change within the schema.

In the state of the model extraction we have been discussing in this section, there are no migrations left in `AppComponent`. Both `Team` and `Game` were extracted from it in the previous chapter. If, for example, `Game` were still in `AppComponent`, instead of changing the migration, what we would want to change is the resulting name of the table that is created. Without changing the existing migration, we achieve this by adding a new migration that renames the table from `app_component_games` to `web_ui_games`.

## Hypothetical new migration to rename database tables affected by component rename

[Click here to view code image](#)

```
1 class RenameNamespaceOfGames < ActiveRecord::Migration
2   def change
3     rename_table :app_component_games, :web_ui_games
4   end
5 end
```

This illustrates that the mechanics handling migrations are the same between renaming a component and moving a model from one component to another.

### 4.6.3 Refactoring a Component Name Throughout the Application

The preceding steps refactor the internals of the component, but not yet any consumer of it. Within CBRA, one observation greatly simplifies how one thinks about ensuring that the corresponding name changes occur throughout the application (i.e., not just the component works after the name change, but also the entire application does): The exact same rules that apply to refactoring the component apply to refactoring references to it (i.e., other parts of the application). Generally, there should be significantly fewer mentions of the component name outside of it, but where they exist, the rules around renaming are the same.

### 4.6.4 An Even More Mechanical Approach

This refactoring is probably the most mechanical of all the refactorings discussed so far. Indeed, it is so mechanical that, in many programming languages, it is easy to implement as an automatic refactoring. Due to Ruby's rich syntax and metaprogramming capabilities, it is very difficult to do the same for components. However, if the refactoring originates from a component name that has no homonyms and there is no metaprogramming that creates unforeseen consequences, one can build a tool to automatically refactor the name of a component.

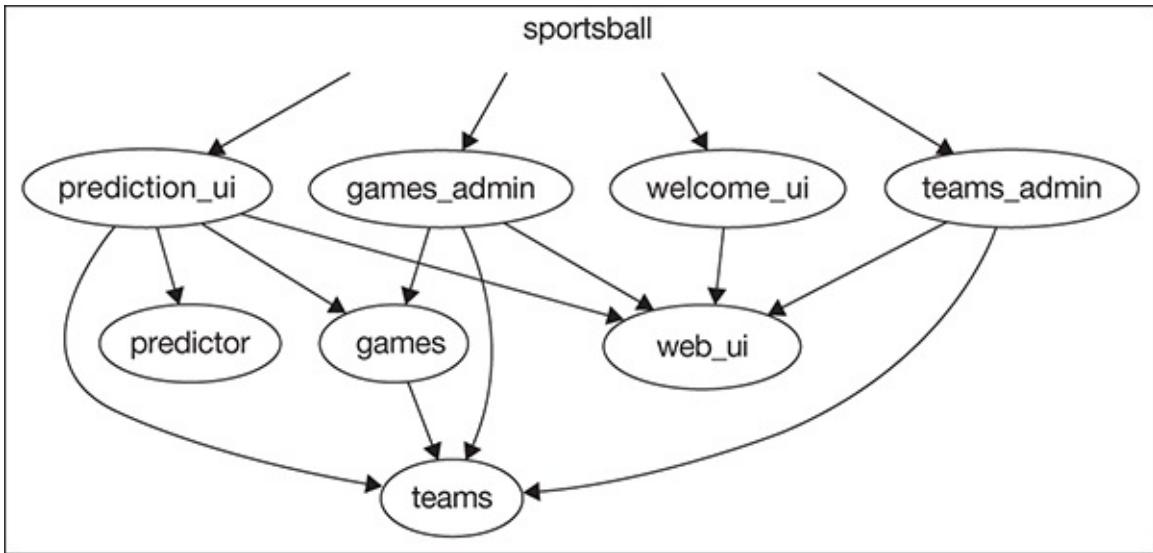
Enter `cbratools` (<https://github.com/shageman/cbratools>), a gem that does exactly that: It can rename a component and the entire CBRA application to make the refactoring of component names as painless as possible.

**cbratools renames a component (results omitted). Execute in ./**

[Click here to view code image](#)

```
$ gem install cbratools
$ rnc AppComponent WebUi
  #renames the component
$ rnm AppComponent WebUi db/migrate db/schema.rb
  #creates needed migrations to support the component name change
```

After we rename `AppComponent` into `WebUi`, the app's dependency diagram looks as in [Figure 4.17](#).



**Figure 4.17.** Sportsball with WebUI dependency

## 4.7 More Component Refactoring Patterns

With the component-based Rails patterns and refactorings we have seen so far, we were able to transform a standard Rails application into a very different, well-structured, component-based application. The refactorings discussed so far are:

1. Move an entire app into a component—Chapter 2
2. Extract a domain gem—[Section 4.2](#)
3. Pull up a UI component—[Section 4.4](#)
4. Push down a model component—[Section 4.5](#)
5. Rename a component—[Section 4.6](#)

We have already seen that the types of activities needed for these refactorings are very similar:

- Moving files
- Renaming namespaces
- Adding and removing gems
- Renaming database table prefixes

In a way, these refactorings are representative of all the refactorings that can occur in a component-based Rails application. The first one, moving an entire app into a component, stands out as an anti-pattern of componentization itself, but is a very helpful tool to introduce all the pitfalls that occur during component development. The next four describe three basic *movements* that a refactoring can initiate:

1. down (domain gem and model component extraction)
2. up (UI component extraction)
3. in-place (component renaming)

One possible movement was not covered: sideways. That is because there are no new activities to introduce in a sideways separation of components.

This section introduces the splitting of a component as the “sideways” refactoring. It will also introduce several other refactorings that are important in day-to-day work with component-based applications.

#### 4.7.1 Splitting One Component in Two: Disentangling Unrelated Concepts

At this point, every Sportsball component contains only one thing: one controller, one model, one class. If the app continues to grow, more things will be added to these components. And, at some point, we might find that a component has grown too large. What is too large? When it contains two unrelated concepts. Why? Because, what *could* be separated *should* be separated. If it is not separated, it likely will become entangled sooner or later and take a much more difficult refactoring to separate out.

As discussed before, it is the context of the component given by its name and the component’s role in the application that creates the boundaries for what should go into it and what should not.

If two unrelated concepts are discovered to be within one component, there are two appropriate engineering responses: either change the name of the component to reflect the multiple concepts within it or split the component to ensure every component encapsulates only one concept.

#### 4.7.2 API Component

API components are a typical occurrence in applications that serve both a web user interface as well as an API for mobile apps or third-party clients. An API component is a component of controllers and presenters exclusively used within the context of the API.

ActionPack supports controllers that serve up data for multiple formats, like HTML, JSON, and XML. This can be used to serve up an API from the same controller as a web page. As long as the two destinations serve the same purpose, this can work reasonably well. For example, the web site shows data in the same way as the mobile application using the API. However, this does not work well if there is a difference in use—for example, when the web page is an administrative interface and the API is not intended for admin purposes. My experience is that the latter is almost always true, or it becomes true very quickly over the course of development.

When API and Web deviate, they will have separate controllers. Applying the top-down component identification technique from [Section 4.3](#)—when parts of the app are distinct and have different uses—they can be split in two.

Controllers generally tend to be very well-structured. They depend on ApplicationController or some subclass thereof. Two individual controllers, however, usually do not depend on each other. Because of this, an API component extraction is a straightforward, sideways extraction. During the refactoring, it is possible that some existing base controllers need to move into a common base component. Complete separation of the API component from other UI components is often possible

because commonly shared aspects, like authentication, are implemented differently.

#### 4.7.3 Third-Party Service Adapter

These days, SaaS like S3, braintree, or twillio typically come with a gem that is made to make the use of the service's API as painless as possible. Such gems typically require an initializer to configure settings and secrets. Once that is done, access to the API is a question of calling the right methods on the gem's classes and modules.

The pattern for using these gems in a component-based application is to wrap the access to this third-party gem in a CBRA gem. The intent is to reduce the exposure of service-related surface area for most of the parts of the application.

Just like we discussed before in regard to the lack of intent in the methods offered by ActiveRecord, the methods offered by third-party service gems must lack application-specific intent. This is because the intent of the gem developers must be to expose all the methods of the service via the gem, whereas within *our* application we use the gem only to solve a problem specific to *our* use cases.

A third-party service adapter gem solves this problem by wrapping the service provider's generic gem. The wrapper (i.e., adapter) exports only those methods that are needed by the application. It might use different method names and signatures. It can combine methods to aggregates that make sense in the context of the application. It will not expose most of the methods available in the third-party gem.

The preceding changes expose the adapted gem's functionality in an application-specific, meaningful way. The adapter gem also ensures that the vast majority of the application is (or *can* be unaware) that a third-party service is used at all. For example, if braintree (<https://braintree.com>) is used to process payments, using its gem without such an adapter will litter the constant Braintree throughout the application. A service adapter gem ensures that Braintree is replaced with PaymentProvider.

Together with an application-specific method interface, a more context-specific name like PaymentProvider sets the application up for changeability. If a different payment provider were to turn out to be a better fit for the company in the future, the change would be confined to the adapter gem.

How would one go about changing payment providers? With a third-party service adapter gem, it is really straightforward:

1. Duplicate the payment provider adapter gem folder (give the folder a different name than the original for now).
2. Remove the gem dependency on the original payment provider and add the dependency on the new payment provider.
3. Run the tests. They should all fail.
4. Fix the tests by replacing the uses of the original payment provider by the analog uses of the new payment provider.
5. Once all tests are green: Delete the original adapter gem folder and rename the folder

of the new adapter gem to have the same name.

6. Run all tests. If the gem was fully tested, there should be (almost) no changes in other components or the main app.

#### 4.7.4 Common Functionality Component

While the name “common functionality component” is very clumsy, it is an attempt to capture the original intent of gems in the Ruby ecosystem. Gems were created to encapsulate functionality that would likely be reused in other contexts. To date, rubygems (<https://rubygems.org>) has served up billions of requests, with every one adding a gem offering reusable functionality to applications around the world.

What are you looking for? Pagination? Kaminari! Administration? ActiveAdmin! Security? ActsAsParanoid! And so forth. You get the idea.

Many gems were originally written by engineers who realized that in different applications, they wrote the same code again and again. In component-based applications, this can happen within one application. That is because, in a component-based application, it is impossible to reuse functionality from other components that a component does not depend upon. Either common code will be duplicated or dependencies will be added for the code to be reused. Many of the refactoring techniques discussed up to this point are aimed at driving out unnecessary dependencies.

There are a number of components I have found on previous projects. One is `soft-deletable` (that is, *our* project’s flavor of it). `Retierable` was a concept and gem that allows for lookup table entries to be put out of use without breaking read access and without forcing updates of all records. `AbstractDomainModel` was a small gem that allowed data transfer objects (<https://martinfowler.com/eaaCatalog/dataTransferObject.html>) to share a small set of characteristics defining their basic properties, like identity, equality, and validity.

Common functionality components are typically dependent upon, directly and indirectly, by many if not most of the components in an application.

Common functionality components are exactly the types of gems found on [rubygems.org](https://rubygems.org). Despite this, the teams that created these components never published any versions of the gems from the component-based apps—with widely varying reasons. One common problem is that the company’s intellectual property policy does not have an easy process for open-sourcing any code. Even if allowed, having to support and maintain a gem publicly takes time and effort, which a team may decide to invest. Also, once published, and if the gem is picked up by others, community input will change aspects of the gem and its API, which can distance the gem from its use on the originating project.

---

a. <https://semver.org/>

# CHAPTER 5

---

## From Ball of Mud to First Components



Photo: Roby1960/Shutterstock

This chapter is all about starting with component-based principles in *existing applications*. As such, it is complementary to [Chapter 2](#) in which we discussed how to start with component-based applications *from scratch*. All the techniques described in this chapter are applicable to applications of arbitrary size. The only hard requirement for most of them is that the application uses Rails 3 or higher.

### 5.1 Small Steps

The good news regarding small steps to introduce components is that the previous chapters have introduced all the building blocks for this. The following refactorings can be applied even in the biggest ball of mud:

- Extract domain gem: [Section 4.2](#)
- Push down model component: [Section 4.5](#)
- Extract third-party service adapter: [Section 4.7.3](#)

- Extract common functionality component: [Section 4.7.4](#)

The commonality among these refactorings is that they push code *down* into new components. That means that the main application will depend on these new components and the components don't need to depend on any part of the existing system. This is critical, as there is no way for a gem to depend on code in an application (without some serious hacking).

Except for the model component, all of these refactorings will also lead to gems (not Rails engines) being extracted. That means the other three refactorings are possible with any version of Rails, even Rails 2.

The model component extraction is likely a big undertaking; in a big old ball-of-mud application, there are likely not many seams that allow for an easy extraction of one or more ActiveRecord models. The biggest chance of that statement being untrue is with models at the fringe of the domain of the application. Print an entity–relationship diagram using a tool like Rails ERD (<http://rails-erd.rubyforge.org/>) and visually search for unconnected or less entangled clusters of tables as a starting point.

If there is any issue with these small step refactorings, it is that they tend to chip away at the problem of the big ball of mud instead of tackling it head-on. In effect, it is unlikely for these refactorings to change the large-scale landscape of the application.

And that's why we have the big steps.

## 5.2 One Big Step

The small step refactorings are not creating big enough changes in an application such that all the possible component-based refactorings discussed in [Chapter 4](#) become available. That means if components only ever get pushed *down* (and these components tend to be small), then we can never apply any *sideways* or *upward* directed refactorings to the vast majority of code in the application. That is, we cannot pull up a UI component, as it likely can't depend on everything it needs because most of it is stuck in the main application. Consequently, we also cannot disentangle two concepts via a sideways extraction, since most concepts are present directly in the application and not in a component.

The big step that answers this problem is moving almost all the application into a component. That means moving all models and anything that won't "come loose" (i.e., anything that the models require to get their tests to pass). While not reducing the complexity of any part of the application, it immediately exposes that code to all possible refactorings and thus enables a much broader refactoring much more quickly than the small steps ever could.

To adequately show how this process works, we need to switch example applications. Sportsball is in no way near the complexity of any application of some age and complexity we would find in the wild, and it will not exhibit the common issues that arise during this refactoring. Instead we are using the `ticketee` sample application found among the examples used in the book *Rails 4 in Action* ([https://github.com/rubysherpas/r4ia\\_examples](https://github.com/rubysherpas/r4ia_examples)), but in a forked version ([https://github.com/shageman/r4ia\\_examples](https://github.com/shageman/r4ia_examples)) that adds (very) basic tests for the models.

### 5.2.1 Prerequisites

We are actually *not* going to look at `what ticketee does` during any part of this process. The refactoring we are applying is purely mechanical and doesn't require that knowledge. This means that the refactoring is applicable to any Rails codebase, irrespective of domain or size. As with every refactoring discussed so far, ample tests that verify the correct behavior of the system are a necessity to make this refactoring work reliably.

### 5.2.2 Script This!

During this refactoring, in every codebase of some size there is a chance for unforeseen difficulties and complications that take time to resolve. Especially when working on an application with a couple of people or multiple teams, this poses a significant problem. If it takes any significant amount of time, a refactoring like this quickly becomes impossible to complete as the amount of code that is touched and moved is so large.

To prevent problems with merge conflicts, I recommend scripting the refactoring instead of manually working through it. Scripting it allows us to test out the necessary changes repeatedly and thus iterate on the refactoring. Every run of the refactoring script is fast, which means this process also doesn't really slow down the refactoring process itself. Between iterations of the refactoring, the latest code changes can continually be pulled into the local codebase so that the refactoring is always tested against the latest code. Finally, when the refactoring is completed, the last pull of the current source code is followed by one last run of the refactoring, after which all code can go back into source control. This makes it feasible to coordinate the run of this refactoring between teams by setting a very short "stop all work" period, during which the refactoring can be applied without conflicts.

### 5.2.3 A Scripted Refactoring

So, how do we start this?

#### **persistence extraction script**

[Click here to view code image](#)

```
1 #!/bin/bash --login
2
3 ensure() {
4     "$@" || exit 1
5 }
6
7 cd r4ia_examples/ticketee;
8
9 git checkout . && git clean -fd
10
11 rvm use 2.2.1@r4ia --create
12
13 gem install bundler
14
15 bundle
```

The ensure function defined in lines 3 through 5 is a helper that will be used in conjunction with test runs to ensure that if a particular test run fails, the scripts exit right away. Line 7 reveals a very interesting aspect of the scripting of this refactoring: The refactoring script is kept *outside* of the root of version control. This is so line 9 can wipe out any changes from previous runs of the script without wiping out the refactoring script itself. If you want to version control the refactoring script, create a new repository specifically for it. Next, lines 11 through 15 ensure that we are using a known version of Ruby (version 2.2.1 in this case) and gemset (with the name r4ia), and that bundler is installed and has run. We thus know all our external dependencies are satisfied.

## persistence extraction script (continued)

[Click here to view code image](#)

```
17 #####
18 ### CREATE PERSISTENCE COMPONENT AND MOVE CODE
19 #####
20
21 rails plugin new components/persistence \
22   --full \
23   --skip-bundle --skip-git \
24   --skip-test-unit \
25   --skip-action-view --skip-sprockets --skip-javascript \
26   --dummy-path=spec/dummy
27
28 mv app/models/* components/persistence/app/models
29 mkdir -p components/persistence/spec/models
30 mv spec/models/* components/persistence/spec/models
31
32 #pushd components/persistence && ensure eval rspec && popd
33
34 cp spec/spec_helper.rb components/persistence/spec/spec_helper.rb
35 cp spec/rails_helper.rb components/persistence/spec/rails_helper.rb
36
37 #pushd components/persistence && ensure eval rspec && popd
38
39 echo 'ENV["RAILS_ENV"] ||= "test"
40 require "spec_helper"
41 require File.expand_path("../dummy/config/environment", __FILE__)
42 require "rspec/rails"
43
44 Dir[Rails.root.join("spec/support/**/*.rb")].each { |f| require f }
45
46 ActiveRecord::Migration.maintain_test_schema!
47
48 RSpec.configure do |config|
49   config.fixture_path = "#{::Rails.root}/spec/fixtures"
50
51   config.use_transactional_fixtures = false
52
53   config.infer_spec_type_from_file_location!
54
55   config.include Warden::Test::Helpers, type: :feature
56   config.after(type: :feature) { Warden.test_reset! }
57   config.include Devise::TestHelpers, type: :controller
```

```
58 end
59 ' > components/persistence/spec/rails_helper.rb
60
61 #pushd components/persistence && ensure eval rspec && popd
```

With the previous code snippet, the refactoring begins in earnest. Lines 21 through 26 create a new component, which is called `persistence`. Although `ball_of_mud` will often be a much better fitting component name, it is rude to the engineers of the codebase. I defend `persistence` as an *aspiring* component name that, hopefully, will fit equally perfectly in the future. Either that, or we manage to refactor it out of existence by continuously removing code from it.

Lines 28 through 35 copy the core of `persistence` into place—the core being the models (line 28) and the model specs (lines 29 and 30). Lines 34 and 35 copy the spec helpers used in the main application into the component. The idea behind this is to start with what already works in one place and adapt it only where something needs to be changed to work within a component. Lines 39 through 59 perform the first such adaptation. Within the script, we echo the contents of the file we are trying to write (in this case `rails_helper.rb`) and use `>` to write the text to the actual file (see line 59). Without it, the error message of running tests is the following:

## Error when running the tests from line 37

[Click here to view code image](#)

```
./ticketee/components/persistence ./ticketee
./ticketee/components/persistence/spec/rails_helper.rb:4:in `require':
  cannot load such file --
./ticketee/components/persistence/config/environment (LoadError)
```

The adaptation necessary in this case is where to find `environment.rb` (line 3 of `rails_helper.rb`). In a Rails application, it is typically `require File.expand_path('../config/environment', __FILE__)`, but in a component, due to the engine's dummy application, `require File.expand_path("../dummy/config/environment", __FILE__)` is needed.

Lines 32, 37, and 61 each contain the following code: `pushd components/persistence && ensure eval rspec && popd`. Uncommented, these lines test the state of the refactoring at that spot in the script. Spread these tests around your version of the script as long as it is helpful. When an early part of the script feels like it is becoming stable, comment out the line so as not to slow down the script unnecessarily.

## Error when running the tests from line 61

[Click here to view code image](#)

```
./ticketee/components/persistence/spec/dummy/db/schema.rb
  doesn't exist yet.
Run `rake db:migrate` to create it, then try again. If you do not
intend to use a database, you should instead alter
./ticketee/components/persistence/spec/dummy/config/application.rb
to limit the frameworks that will be loaded.
```

```
./ticketee/components/persistence/app/models/attachment.rb:4:  
  in `<class:Attachment>': uninitialized constant  
    Attachment::AttachmentUploader (NameError)
```

This error showcases perfectly how to proceed through this refactoring. It highlights two separate problems: first, that we haven't yet moved any migrations, and second, that we don't have access to `AttachmentUploader` in the context of the component. We will need to tackle each of these problems in turn.

The migrations should be moved, because all the models were moved. Since it is *all* the models, it is also *all* the migrations.

## **persistence extraction script (continued)**

[Click here to view code image](#)

```
63 mkdir -p components/persistence/db/migrate  
64 mv db/migrate/* components/persistence/db/migrate  
65  
66 #pushd components/persistence && \  
67 # rake db:migrate RAILS_ENV=test && ensure eval rspec && \  
68 # popd
```

The solution to the missing `AttachmentUploader` is a bit more complicated. The first observation is that, despite the error message saying the missing constant is `Attachment::AttachmentUploader`, the problem is in fact that `AttachmentUploader`, defined in `app/uploaders`, was not moved into the component.

This is the kind of stuff that makes the name *ball of mud* so appropriate. Why is it that an uploader to a model is known by the model itself? It should make sense that the uploader needs to know about the models it can upload, but it is the other way around: The model knows that it can be uploaded by a particular uploader. This is an unfortunate entanglement. In this case, the entanglement stems from the use of the CarrierWave uploader, which has made this design choice that I fundamentally disagree with; see CarrierWave ActiveRecord API (<https://github.com/carrierwaveuploader/carrierwave#activerecord>).

Attempting to fix this issue by removing the cause would distract us from the ultimate goal of the refactoring and, most importantly, require us to make nonmechanical changes (i.e., we'd have to think through which change we would like to see instead of moving forward with extracting our first component).

If we don't remove the entanglement, we must pull the uploader into `persistence` as well.

## **persistence extraction script (continued)**

[Click here to view code image](#)

```
70 mkdir -p components/persistence/app/uploaders  
71 mv app/uploaders/* components/persistence/app/uploaders  
72  
73 #pushd components/persistence && \  
74 # rake db:migrate RAILS_ENV=test && ensure eval rspec && \  
75 # popd
```

## Error when running the tests from line 73

[Click here to view code image](#)

```
~/.rvm/gems/ruby-2.2.1@r4ia/gems/activerecord-4.2.1/
  lib/active_record/dynamic_matchers.rb:26:in `method_missing':
undefined method `devise' for #<Class:0x007fe8104c13c8>
  (NoMethodError)
```

The method `devise` that is missing here comes from the `Devise` gem (as we can confirm by checking the original `gemfile`). So, in the next adaptation we add `s.add_dependency "devise", "3.4.1"` to `persistence.gemspec`.

## persistence extraction script (continued)

[Click here to view code image](#)

```
77 echo '
78 $:.push File.expand_path("../lib", __FILE__)
79
80 # Maintain your gems version:
81 require "persistence/version"
82
83 # Describe your gem and declare its dependencies:
84 Gem::Specification.new do |s|
85   s.name      = "persistence"
86   s.version   = Persistence::VERSION
87   s.authors   = ["TODO: Write your name"]
88   s.email     = ["TODO: Write your email address"]
89   s.homepage  = "TODO"
90   s.summary   = "TODO: Summary of Persistence."
91   s.description = "TODO: Description of Persistence."
92   s.license   = "MIT"
93
94   s.files = Dir["{app,config,db,lib}/**/*",
95                 "MIT-LICENSE", "Rakefile", "README.rdoc"]
96
97   s.add_dependency "rails", "4.2.1"
98   s.add_dependency "devise", "3.4.1"
99
100  s.add_development_dependency "sqlite3"
101  s.add_development_dependency "rspec-rails", "3.2.1"
102 end
103 ' > components/persistence/persistence.gemspec
104
105 #pushd components/persistence && \
106 # rake db:migrate RAILS_ENV=test && ensure eval rspec && \
107 # popd
```

## Error when running the tests from line 105

[Click here to view code image](#)

```
./ticketee/components/persistence/spec/rails_helper.rb:17:
  in `block in <top (required)>': uninitialized
  constant Warden (NameError)
```

The error message following this is a bit confusing and does not immediately point to the necessary next step. However, from all our previous refactorings we know that, in gems,

dependencies should be required explicitly. Thus, we add require "devise" to persistence.rb.

## persistence extraction script (continued)

[Click here to view code image](#)

```
109 echo '  
110 require "devise"  
111  
112 module Persistence  
113   require "persistence/engine"  
114 end  
115 ' > components/persistence/lib/persistence.rb  
116  
117 #pushd components/persistence && \  
118 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \  
119 #  popd
```

### Error when running the tests from line 117

[Click here to view code image](#)

```
./ticketee/components/persistence/app/uploaders/  
attachment_uploader.rb:3: in `<top (required)>': uninitialized  
constant CarrierWave (NameError)
```

Having moved AttachmentUploader into persistence, we should have expected that CarrierWave would need to be required as a gem dependency. We are finally getting this error. Adding carrierwave to the gemspec and to persistence.rb should fix it.

---

## What if Multiple Updates to a File Are Necessary?

Note that this is the second time in this script that we are changing persistence.gemspec. In this, I am simply adding another echo of the new file's contents and writing to the file again. While that may seem unnecessary—we could, after all, go back up in the script and modify the original change—I recommend doing this as long as the sheer length of the refactoring script file isn't becoming a nuisance. This will keep the steps of the refactoring linear.

If the file feels like it is growing too long, try the following: In a refactoring support directory, recreate the folder structure of the files that need to be changed and place the needed version of the file in there. Use find and cp to copy all files from the support directory into the refactored application at the start of the refactoring script, which now only creates the component and moves needed files.

---

## persistence extraction script (continued)

[Click here to view code image](#)

```
121 echo '  
122 $:.push File.expand_path("../lib", __FILE__)  
123  
124 # Maintain your gems version:  
125 require "persistence/version"
```

```

126
127 # Describe your gem and declare its dependencies:
128 Gem::Specification.new do |s|
129   s.name          = "persistence"
130   s.version       = Persistence::VERSION
131   s.authors       = ["TODO: Write your name"]
132   s.email         = ["TODO: Write your email address"]
133   s.homepage      = "TODO"
134   s.summary        = "TODO: Summary of Persistence."
135   s.description    = "TODO: Description of Persistence."
136   s.license        = "MIT"
137
138   s.files = Dir["{app,config,db,lib}/**/*",
139                 "MIT-LICENSE", "Rakefile", "README.rdoc"]
140
141   s.add_dependency "rails", "4.2.1"
142   s.add_dependency "devise", "3.4.1"
143   s.add_dependency "carrierwave", "0.10.0"
144
145   s.add_development_dependency "sqlite3"
146   s.add_development_dependency "rspec-rails", "3.2.1"
147 end
148 ' > components/persistence/persistence.gemspec
149
150 echo '
151 require "devise"
152 require "carrierwave"
153
154 module Persistence
155   require "persistence/engine"
156 end
157 ' > components/persistence/lib/persistence.rb
158
159 #pushd components/persistence && \
160 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \
161 #  popd

```

## Error when running the tests from line 159

[Click here to view code image](#)

```

~/.rvm/gems/ruby-2.2.1@r4ia/gems/activerecord-4.2.1/lib/active_record
/dynamic_matchers.rb:26:in `method_missing':
undefined method `searcher' for #<Class:0x007ff634144358> (NoMethodError)

```

Now a searcher method is missing. A search through the codebase reveals that a searcher gem is used by the application. Because it is pulled from a Github location, in addition to the usual changes we need to also adapt Persistence's Gemfile.

Note that because searcher is installed through git, we need to add a separate bundle step to the running of the tests in line 212.

## persistence extraction script (continued)

[Click here to view code image](#)

```

163 echo '
164 $:.push File.expand_path("../lib", __FILE__)

```

```

165
166 # Maintain your gems version:
167 require "persistence/version"
168
169 # Describe your gem and declare its dependencies:
170 Gem::Specification.new do |s|
171   s.name          = "persistence"
172   s.version       = Persistence::VERSION
173   s.authors       = ["TODO: Write your name"]
174   s.email         = ["TODO: Write your email address"]
175   s.homepage      = "TODO"
176   s.summary        = "TODO: Summary of Persistence."
177   s.description    = "TODO: Description of Persistence."
178   s.license        = "MIT"
179
180   s.files = Dir["{app,config,db,lib}/**/*", "MIT-LICENSE",
181                 "Rakefile", "README.rdoc"]
182
183   s.add_dependency "rails", "4.2.1"
184   s.add_dependency "devise", "3.4.1"
185   s.add_dependency "carrierwave", "0.10.0"
186   s.add_dependency "searcher"
187
188   s.add_development_dependency "sqlite3"
189   s.add_development_dependency "rspec-rails", "3.2.1"
190 end
191 ' > components/persistence/persistence.gemspec
192
193 echo '
194 source "https://rubygems.org"
195
196 gemspec
197
198 gem "searcher", github: "radar/searcher",
199   ref: "c2975124e11677825481ced9539f16f0cb0640de"
200 ' > components/persistence/Gemfile
201
202 echo '
203 require "devise"
204 require "carrierwave"
205 require "searcher"
206
207 module Persistence
208   require "persistence/engine"
209 end
210 ' > components/persistence/lib/persistence.rb
211
212 #pushd components/persistence && bundle && \
213 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \
214 #  popd

```

## Error when running the tests from line 212

[Click here to view code image](#)

```
~/.rvm/gems/ruby-2.2.1@r4ia/gems/activerecord-4.2.1/
  lib/active_record/dynamic_matchers.rb:26:in `method_missing':
```

```
undefined method `devise' for  
#<Class:0x007fe8104c13c8> (NoMethodError)
```

It is pretty surprising that, after all this, we are back to the error of the `devise` method missing. This is proof of the mechanical nature of the refactoring—we didn't move, copy, or change anything that errors or failing tests didn't prove to be needed.

If you have ever worked with Devise, you know that it requires an initializer to be run that configures it. The next lines try out whether that is indeed the problem and move the `devise.rb` initializer into `persistence`, which does in fact solve the problem and fixes `persistence`'s tests.

## **persistence extraction script (continued)**

[Click here to view code image](#)

```
213 mkdir -p components/persistence/config/initializers  
214 mv config/initializers/devise.rb \  
215     components/persistence/config/initializers  
216  
217 #pushd components/persistence && bundle && \  
218 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \  
219 #  popd
```

### **No more error when running the tests from line 217**

[Click here to view code image](#)

```
Finished in 23.55 seconds (files took 4.48 seconds to load)  
140 examples, 0 failures  
  
SUCCESS
```

## 5.2.4 Cleaning Up Persistence

`persistence` is fully functional. A remaining housekeeping chore is the deletion of obsolete directories and files.

## **persistence extraction script (continued)**

[Click here to view code image](#)

```
225 rm -rf components/persistence/app/assets  
226 rm -rf components/persistence/app/controllers  
227 rm -rf components/persistence/app/helpers  
228 rm -rf components/persistence/app/mailers  
229 rm -rf components/persistence/app/models/concerns  
230 rm -rf components/persistence/app/models/.keep  
231 rm -rf components/persistence/app/views  
232 rm -rf components/persistence/lib/tasks
```

## 5.2.5 Using Persistence

At this point in the refactoring, we are back to known territory. We have a new component and want to use it in the main application.

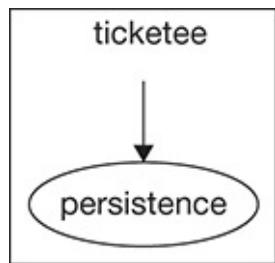
All the last steps, which I am omitting from the text but which are in the refactoring file,

we have previously discussed:

- Changing `engine.rb` to append its migrations
- Entering real values into all `FIXME` and `TODO` spots in `persistence.gemspec` to make it valid
- Loading `persistence` in the main app's `Gemfile`
- Adding `test.sh` files to `persistence` as well as the main app
- Adding `build.sh` to the root of the application

## 5.2.6 Summary

Getting started with CBRA in an existing application can seem like a daunting task. The refactoring in this chapter attempts to make it feasible by mechanizing every step necessary. By making the refactoring scripted, we can iterate on it and develop largely in parallel to possibly ongoing feature development. The result of our efforts, shown in [Figure 5.1](#), may not look like much, but is a huge step toward a better-structured app.



**Figure 5.1.** Component structure of `ticketee` after the persistence extraction

Again, it is crucial to avoid trying to solve the problems of the codebase during this refactoring. In the spirit of test-driven development's "doing the simplest thing that could possibly work," refactor aggressively toward the extraction of the first component. If that feels like creating a worse application, remind yourself that you are starting with *one* ball of mud and ending with *two!* That must mean that the size of the mud ball is smaller afterward. Hence, without doing anything else, you have improved the codebase!

# CHAPTER 6

## Component-Based Rails in Relation to Other Patterns



Photo: Dr Ajay Kumar Singh/Shutterstock

Component-based development can be combined with other approaches to making applications more maintainable. That is because where architectural approaches add organizational ideas and idioms help organize a codebase, components add rigor to the content, the boundaries, and the relationships of those idioms.

It has taken a long time for components implemented via gems (and for Rails via engines) to be picked up by the community at large. That is why the previous chapters focused on the technical aspect and the mechanics of the pattern. In this chapter, we turn to the question of how CBRA fits in with those other approaches. We'll see that, when combined, we can lift architectural patterns for Rails to a whole new level.

A lot is being written about architecture. Architecture for Rails specifically is no exception. The selection presented here is mainly derived based on the attention the various approaches are receiving. I am not trying to achieve here a discussion of the merits of all these approaches, nor am I trying to say that using one or the other would be good

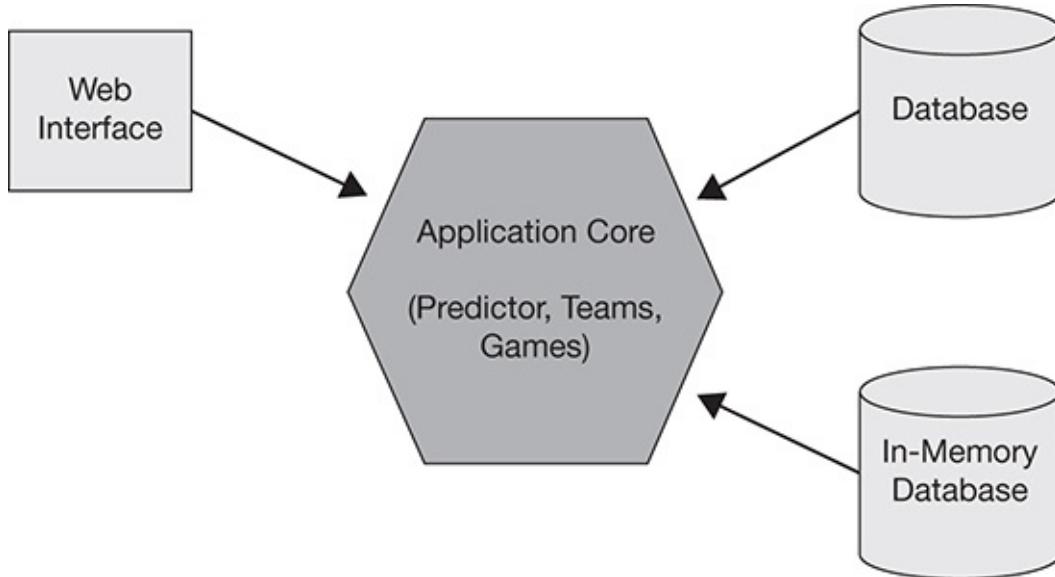
for your application. That is beyond the scope of this book.

## 6.1 Hexagonal Architecture

Hexagonal Architecture was created by Alistair Cockburn as an attempt at an architecture that can combat the proliferation of business logic into all parts (and specifically the interface) of applications (<http://alistair.cockburn.us/Hexagonal+architecture>). Where commonly another “layer” is introduced, Hexagonal Architecture introduces an architecture based on ports and adapters. The ports create entry and exit points for user input and application output. Adapters embody the connection to the actual thing on the outside of the core of the application.

The image of the hexagon is intended to evoke the possibility of multiple ports and adapters. This contrasts with the idea of layers, which allows for one interface—the top—and one “backend” (most commonly perhaps a database) the bottom. It thus tries to capture in one image that there might be more than a Web interface providing inputs and displaying outputs. There might also be more than a database that persists data. For example, in tests, the application could be driven by scripts instead of the Web interface, and the database may be replaced with a mock.

Figure 6.1 shows a high-level example of a Hexagonal Architecture. The core of the application is the hexagon in the middle. It contains the business logic and thus the meat of the domain as embodied by code. The hexagon is surrounded by the various ways in which it interacts with the outside world.



**Figure 6.1.** Example of a Hexagonal Architecture

### 6.1.1 Hexagonal Architecture and CBRA

The key rule to follow is “that code pertaining to the ‘inside’ part should not leak into the ‘outside’ part” (<http://alistair.cockburn.us/Hexagonal+architecture>). This rule is about boundaries and the direction of dependencies. Within Rails, this rule will be tough to follow without using CBRA. However, using CBRA, the inside will be forced to not depend on the outside by both being implemented as components and by the dependency being from the outside in and not the other way around.

Rails as a framework deals exclusively with the outside of the hexagon. In effect, none of an application’s business logic should depend on the Rails gems. This is impossible if an application is solely implemented within the confines of a “Rails app.” As we will see in the following, it takes a bit of work to make controllers business-logic-less and to create domain models that don’t depend on the framework.

There are numerous ways in which people have implemented the idea of Hexagonal Architecture in Rails. For the following example, I chose to follow the implementation discussed by Graham Ashton in his blog post “Refactoring with Hexagonal Rails” (<https://www.agileplannerapp.com/blog/building-agile-planner/refactoring-with-hexagonal-rails>).

### 6.1.2 Implementing Hexagonal Rails with CBRA

In this discussion, only the “front” of game prediction and the “back” of teams administration is changed to implement a Hexagonal Architecture. While the obvious and simple explanation for this is the laziness of the author, there is a subtler but very important explanation as well: It is absolutely viable to implement the Hexagonal pattern for certain parts of the application only.

#### Why Only Partly Implement a Pattern Like Hexagonal Architecture?

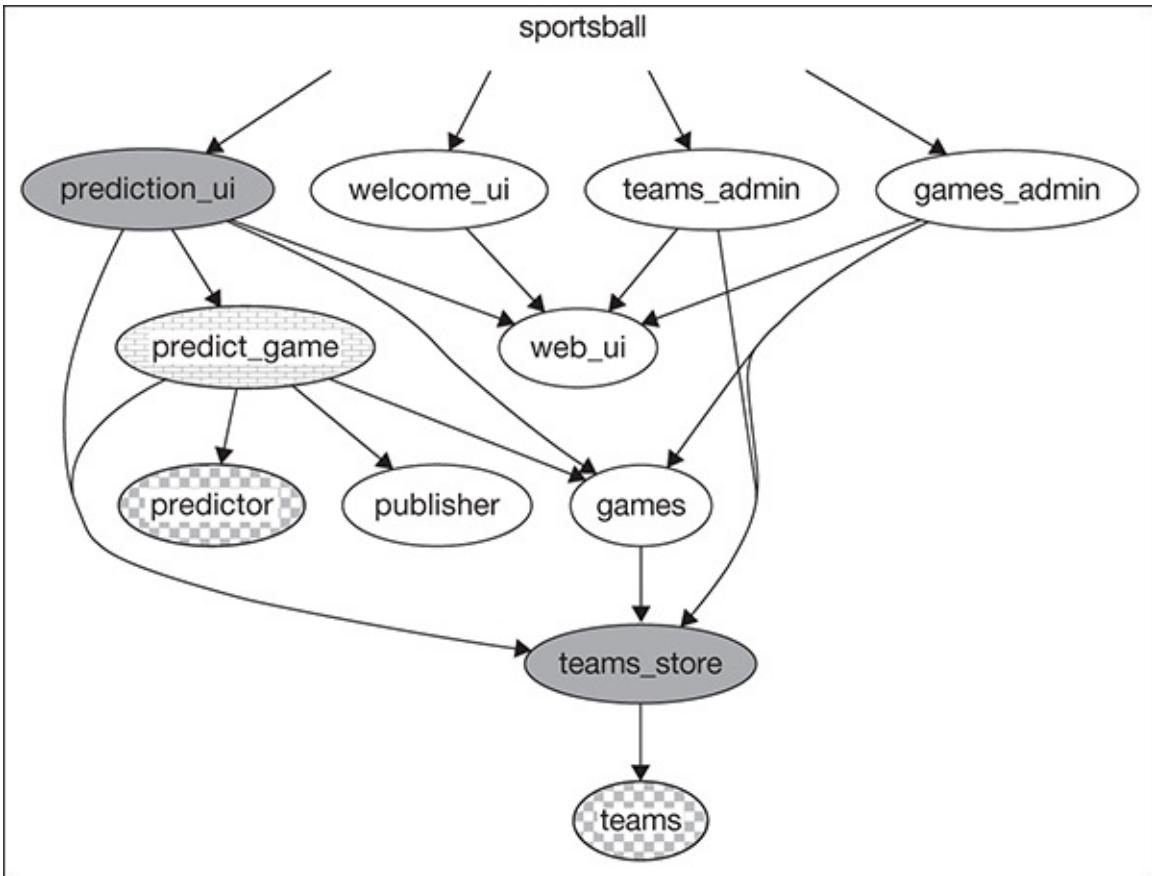
It almost lies in the nature of the architectural patterns that any application that is small enough to be a good explanation of the pattern is too small for the pattern to make any sense.

Patterns are employed as *proven solutions* to *recurring problems*. A small codebase, a small Rails app, has no immediate problems that Hexagonal Rails or CBRA would be needed for. They are too simple. Just take all the code in this book —CBRA is kinda’ ridiculous for all of it!

Only slightly more differentiated, one can say that simple *parts* of codebases are too small for these patterns to make sense. If there is a part of the app that can use a generic ActiveAdmin interface to manage a bunch of models with hardly any app code, it is probably a good idea to leave that in. Instead, apply sophisticated patterns in complex parts of the application. They will create more of a benefit there—a benefit that has a higher chance of outweighing the cost of applying the pattern.

---

The component diagram of this solution looks as shown in [Figure 6.2](#).



**Figure 6.2.** Components of Sportsball’s Hexagonal Architecture

There are two visible additions in comparison to the last version of the application from [Section 4.6](#): Publisher and PredictGame. Both components are involved in handling the “front” of game prediction.

The components with checkerboard fill, predictor and game, are on the inside of the hexagon. The brick-patterned predict\_game sits in the shell of the hexagon and encapsulates the business logic of predicting games. The solid-filled prediction\_ui and teams\_store sit outside the hexagon, in the front and back, respectively.

### 6.1.3 Teasing Out an Adapter in the Frontend

When checking out the existing PredictionsController from [Section 4.6](#), one may notice a small problem for showing this refactoring: It does not contain any business logic.

`./components/prediction_ui/app/controllers/predictions_controller.rb` from [Section 4.6](#)

[Click here to view code image](#)

```

1 require_dependency "prediction_ui/application_controller"
2 module PredictionUi
3   class PredictionsController < ApplicationController
4     def new
5       @teams = Team.all
6     end
7
8     def create

```

```

9 predictor = Predictor::Predictor.new(Teams::Team.all)
10 predictor.learn(Games::Game.all)
11 @prediction = predictor.predict(
12     Teams::Team.find(params["first_team"]["id"]),
13     Teams::Team.find(params["second_team"]["id"]))
14 end
15 end
16 end

```

Let's say the next feature that our application is supposed to implement is to show a message that a prediction is not possible when the same team is selected twice. The most straightforward implementation is in the `create` method of the controller. It has the IDs of the teams to predict for and can add an `if` statement to decide to show a different message in case they are equal.

This, however, constitutes leakage of business logic into the *outside*. Instead, let's implement this using Hexagonal's ports-and-adapters pattern.

`Publisher` is a new component with a single contribution to the overall application, the `Publisher` module. It is an exact copy from Graham's aforementioned blog post (<https://www.agileplannerapp.com/blog/building-agile-planner/refactoring-with-hexagonal-rails>). Separated into a component, it doesn't depend on any other parts of the application and is tested in isolation.

**`./components/publisher/lib/publisher.rb`**

[Click here to view code image](#)

```

1 module Publisher
2   def add_subscriber(object)
3     @subscribers ||= []
4     @subscribers << object
5   end
6
7   def publish(message, *args)
8     return if @subscribers == [] || @subscribers == nil
9     @subscribers.each do |subscriber|
10       if subscriber.respond_to?(message)
11         subscriber.send(message, *args)
12       end
13     end
14   end
15 end

```

When a class includes `Publisher`, it starts accepting the registration of subscribers via the `add_subscriber` method. That class can then send messages to (i.e., call methods on) all subscribers by calling `publish`.

The class `PredictGame`, in which this is used, lives in the `PredictGame` component —another component with just one thing in it.

**`./components/predict_game/lib/predict_game/predict_game.rb`**

[Click here to view code image](#)

```

1 module PredictGame
2   class PredictGame

```

```

3  include Publisher
4
5  def initialize(teams, games)
6      @predictor = ::Predictor::Predictor.new(teams)
7      @predictor.learn(games)
8  end
9
10 def perform(team1_id, team2_id)
11     if @predictor.game_predictable?(team1_id, team2_id)
12         publish(
13             :prediction_succeeded,
14             @predictor.predict(team1_id, team2_id))
15     else
16         publish(
17             :prediction_failed,
18             @predictor.predict(team1_id, team2_id),
19             "Prediction can not be performed with
20             a team against itself")
21     end
22   end
23 end
24 end

```

PredictGame does a lot of the work that PredictionsController used to do. It gets all teams and all games and sets up the prediction engine in its initialize method. Its perform method performs the prediction and publishes one of two messages: prediction\_succeeded or prediction\_failed.

Finally, PredictionsController now interacts with PredictGame to execute a prediction.

[./components/prediction\\_ui/app/controllers/prediction\\_ui/predict](#)

[Click here to view code image](#)

```

1 module PredictionUi
2   class PredictionsController < ApplicationController
3     def new
4       @teams = TeamsStore::TeamRepository.new.get_all
5     end
6
7     def create
8       game_predictor = PredictGame::PredictGame.new(
9           TeamsStore::TeamRepository.new.get_all,
10          Games::Game.all)
11     game_predictor.add_subscriber(PredictionResponse.new(self))
12     game_predictor.perform(
13         TeamsStore::TeamRepository.new.get(
14             params["first_team"]["id"]),
15             TeamsStore::TeamRepository.new.get(
16                 params["second_team"]["id"]))
17   end
18
19   class PredictionResponse < SimpleDelegator
20     def prediction_succeeded(prediction)
21       render locals: {prediction: prediction, message: nil}

```

```

22     end
23
24     def prediction_failed(prediction, error_message)
25         render locals: {
26             prediction: prediction, message: error_message
27         }
28     end
29   end
30 end
31 end

```

This controller does not know anything about *how* a prediction can succeed or fail. It only knows *that* these two options are possible and has appropriate handler methods implemented in the nested class `PredictionResponse`. The controller thus becomes an *adapter* to the *port* that is defined by `PredictGame`'s interface.

Again, the complexity of this interaction is utterly ridiculous for the problem at hand. `PredictGame`, which includes `Publisher`, makes the decision on predictability and sends messages to a nested class of the controller so that it, as a `DelegateClass` of the controller, can render the correct response to the consumer. What is the net effect on the logic? An `if` statement gets moved from the controller into a non-controller class. However, that `if` statement is business logic and as such does not belong in the controller in a Hexagonal Architecture. One `if` statement. Don't dismiss the pattern just yet, though —it might well be the superior pattern in more complex situations.

#### 6.1.4 A Repository for Data Storage

The attentive reader may have noticed that the way `PredictGame` and `PredictionsController` retrieved teams has changed in the preceding code. No longer are we calling `Teams::Team.all` but rather `TeamsStore::TeamRepository.new.get_all`. That is because of a second port-and-adapter combination employed in this application. This one reimplements the backend for team storage.

Let's look at the most interesting consumer of this change: `TeamsAdmin`'s `TeamsController`:

```
./components/teams_admin/app/controllers/teams_admin/teams_controller.rb
```

[Click here to view code image](#)

```

1 require_dependency "teams_admin/application_controller"
2
3 module TeamsAdmin
4   class TeamsController < ApplicationController
5     before_action :ensure_dependencies
6     before_action :set_team, only: [
7       :show, :edit, :update, :destroy]
8
9     def index
10       @teams = @team_repository.get_all
11     end
12   end

```

```

13  def new
14      @team = Teams::Team.new
15  end
16
17  def edit
18  end
19
20  def create
21      team = Teams::Team.new(team_params[:id],
22          team_params[:name])
23      @team = @team_repository.create(team)
24
25      if @team.persisted?
26          redirect_to teams_teams_url,
27              notice: "Team was successfully created."
28      else
29          render :new
30      end
31  end
32
33  def update
34      if @team_repository.update(@team.id, team_params[:name])
35          redirect_to teams_teams_url,
36              notice: "Team was successfully updated."
37      else
38          render :edit
39      end
40  end
41
42  def destroy
43      @team_repository.delete(@team.id)
44      redirect_to teams_teams_url,
45          notice: "Team was successfully destroyed."
46  end
47
48  private
49  def set_team
50      @team = @team_repository.get(params[:id])
51  end
52
53  def team_params
54      params.require(:teams_team).permit(:name)
55  end
56
57  def ensure_dependencies
58      @team_repository = TeamsStore::TeamRepository.new
59  end
60  end
61 end

```

A new `before_action` in line 5 ensures the controller's dependencies are being loaded. Line 49 reveals that the only dependency of this controller is a new `TeamsStore::TeamRepository`. Throughout the controller, you can see the functionality of this new repository: `create`, `get`, `get_all`, `update`, and `delete`.

What is not yet visible is that this repository is necessary because the Team class was changed to no longer be an ActiveRecord-derived model.

`./components/teams/lib/teams/team.rb`

[Click here to view code image](#)

```
1 module Teams
2   class Team
3     include ActiveRecord::Conversion
4     include ActiveRecord::Validations
5     include ActiveRecord::Naming
6
7     attr_reader :id, :name
8
9     def initialize(id=nil, name=nil)
10      @id = id
11      @name = name
12    end
13
14    def persisted?
15      @id != nil
16    end
17
18    def ==(other)
19      other.is_a?(Teams::Team) && @id == other.id
20    end
21
22    def new_record?
23      !persisted?
24    end
25  end
26 end
```

The repository is necessary because Team no longer has any of the methods it previously got from ActiveRecord::Base that allowed it to create, find, and update itself.

Instead, Team has turned into a value object that can be initialized with :id and :name but that cannot be changed once created. It implements the methods persisted?, ==, and new\_record? as well having the methods it gets from including ActiveRecord::Conversion, ActiveRecord::Validations, and ActiveRecord::Naming. ActiveRecord::Conversion adds conversion methods like to\_param and to\_key. ActiveRecord::Validations adds valid?, its collaborators, and errors. Finally, ActiveRecord::Naming adds the model\_name method to Team.

Team objects are managed through TeamRepository.

`./components/teams_store_db/app/models/teams_store/team_repos`

[Click here to view code image](#)

```
1 module TeamsStore
2   class TeamRepository
3     def get_all
4       TeamRecord.all.map do |team_record|
```

```

5      team_from_record(team_record)
6  end
7 end
8
9 def get(id)
10    team_record = TeamRecord.find_by_id(id)
11    team_from_record(team_record)
12 end
13
14 def create(team)
15    team_record = TeamRecord.create(name: team.name)
16    team_from_record(team_record)
17 end
18
19 def update(id, name)
20    TeamRecord.find_by_id(id).update(name: name)
21 end
22
23 def delete(id)
24    TeamRecord.delete(id)
25 end
26
27 private
28
29 class TeamRecord < ActiveRecord::Base
30   self.table_name = "teams_teams"
31
32   validates :name, presence: true
33 end
34 private_constant(:TeamRecord)
35
36 def team_from_record(team_record)
37   Teams::Team.new(team_record.id, team_record.name)
38 end
39 end
40 end

```

There are two interesting aspects to how `TeamRepository` does its work. First, all its public methods use `TeamRecord` to implement the work of finding and managing database records. To this end, `TeamRecord` is a private, nested class that extends from `ActiveRecord::Base`. It is set up to use the existing `teams_teams` database (line 31). The repository makes sure never to return instances of `TeamRecord` publicly. Instead, it transfers their data into respective `Team` instances via `team_from_record` (line 37) first.

In summary, `Team` is inside the hexagon as a domain model while `TeamRecord` is within the database adapter on the outside.

## 6.1.5 Swappable Data Storage

It remains to be mentioned that all of the preceding files were loaded from `/components/teams_store_db/`, because this version of Sportsball comes with not one but *two* implementations of the teams database: `teams_store_db` and `teams_store_mem`. Next to the gem in `teams_store_db` that implements `Team` and `TeamRepository` as we just saw, there is `teams_store_mem`, which implements an interface-compatible, in-memory team repository.

Let's compare their `gemspec`.

**`./components/teams_store_db/teams_store.gemspec`**

[Click here to view code image](#)

```
1 $:.push File.expand_path("../lib",      _____FILE_____ )
2
3 # Describe your gem and declare its dependencies:
4 Gem::Specification.new do |s|
5   s.name          = "teams_store"
6   s.version       = "0.0.1"
7   s.authors       = ["Your name"]
8   s.email         = ["Your email"]
9   s.summary        = "Summary of TeamsStore."
10  s.description    = "Description of TeamsStore."
11
12 s.files = Dir["{app,config,db,lib}/**/*",
13               "MIT-LICENSE", "Rakefile", "README.rdoc"]
14
15 s.add_dependency "activerecord", "5.1.4"
16 s.add_dependency "teams"
17
18 s.add_development_dependency "rails", "5.1.4"
19 s.add_development_dependency "sqlite3"
20 s.add_development_dependency "rspec-rails"
21 s.add_development_dependency "shoulda-matchers"
22 s.add_development_dependency "database_cleaner"
23 end
```

**`./components/teams_store_mem/teams_store.gemspec`**

[Click here to view code image](#)

```
1 $:.push File.expand_path("../lib",      _____FILE_____ )
2
3 # Describe your gem and declare its dependencies:
4 Gem::Specification.new do |s|
5   s.name          = "teams_store"
6   s.version       = "0.0.1"
7   s.authors       = ["Your name"]
8   s.email         = ["Your email"]
9   s.summary        = "Summary of TeamsStore."
10  s.description    = "Description of TeamsStore."
11
12 s.files = Dir["{app,config,db,lib}/**/*",
13               "MIT-LICENSE", "Rakefile", "README.rdoc"]
14
```

```

15   s.add_dependency "teams"
16
17   s.add_development_dependency "rspec"
18 end

```

The first thing to note is that both gems name their gem `teams_store`. This, and the fact (as we will see) that their interfaces are the same, allows the main application to easily swap back and forth between the two implementations of `TeamsStore`.

The in-memory version of the repository gets away with only requiring `teams` and `rspec`, while the classic, database-based implementation has more dependencies to deal with to make the database connection work.

We start seeing the difference when we look at the implementation of the `TeamStore` repository.

`./components/teams_store_mem/lib/teams_store/team_repository.`

[Click here to view code image](#)

```

1 module TeamsStore
2   class TeamRepository
3     def get_all
4       TeamsStore::Db.get.values
5     end
6
7     def get(key)
8       id = key.to_i
9       TeamsStore::Db.get[id]
10    end
11
12    def create(team)
13      return team if [nil, ""].include? team.name
14
15      id = TeamsStore::Db.get.keys.max &&
16          TeamsStore::Db.get.keys.max + 1 || 1
17      TeamsStore::Db.get[id] = Teams::Team.new(id, team.name)
18    end
19
20    def update(key, name)
21      id = key.to_i
22      return false if [nil, ""].include? name
23
24      TeamsStore::Db.get[id] = Teams::Team.new(id, name)
25      true
26    end
27
28    def delete(key)
29      id = key.to_i
30
31      TeamsStore::Db.get.delete(id)
32      id
33    end
34  end
35 end

```

The repository is backed by the in-memory “database” used: a simple hash.

```
./components/teams_store_mem/lib/teams_store/db.rb
```

[Click here to view code image](#)

```
1 module TeamsStore
2   module Db
3     def self.reset
4       $teams_db = {}
5     end
6
7     def self.get
8       $teams_db
9     end
10    end
11  end
```

Testing within the components is business as usual. The engines implement all the tests for the functionality they control.

More interesting is how the tests for the main application were adapted. Its build script shows that the entire application is fully tested with both implementations of Teams. This is achieved through a symbolic link from `./components/teams_store` to the folder of the respective team implementation to be tested (lines 7 through 8 and 19 through 20 in the following).

```
./build.sh
```

[Click here to view code image](#)

```
1 #!/bin/bash
2
3 result=0
4
5 echo "### TESTING EVERYTHING WITH TEAMS DB"
6
7 rm components/teams_store
8 ln -s teams_store_db components/teams_store
9
10 for test_script in $(find . -name test.sh); do
11   pushd `dirname $test_script` > /dev/null
12   ./test.sh
13   ((result+= $?))
14   popd > /dev/null
15 done
16
17 echo "### TESTING EVERYTHING WITH TEAMS IN MEM"
18
19 rm components/teams_store
20 ln -s teams_store_mem components/teams_store
21
22 for test_script in $(find . -name test.sh); do
23   pushd `dirname $test_script` > /dev/null
24   ./test.sh
25   ((result+= $?))
26   popd > /dev/null
27 done
28
```

```

29 if [ $result -eq 0 ]; then
30     echo "SUCCESS"
31 else
32     echo "FAILURE"
33 fi
34
35 exit $result

```

## 6.1.6 Summary

The ideas and principles of Hexagonal Architecture add more direction to the effort of creating the internal structure of an application. Component-based Rails complements this by adding the rigor that ensures that the pieces only depend on each other in intended ways. CBRA can be used to enforce the directionality of the dependence of the shell on the core.

Again, this is more showcase for the combination of Hexagonal Architecture with CBRA than it was a perfect introduction, let alone implementation, to Hexagonal Architecture.

An interesting aspect that was uncovered is that a Hexagonal Architecture doesn't have to be an all-or-nothing endeavor. The complicated parts (as much as Sportsball has anything complicated) of the application were already written in a way very conducive to a Hexagonal structure. While it may not make sense for the admin (CRUD) part of an application to ever be converted away from “just the Rails way,” other parts may very well benefit significantly.

## 6.2 Data-Context-Integration (DCI)

DCI is a software pattern developed by Trygve Reenskaug and James Coplien as an answer to the gap between our ability to capture what a system *is* versus what it *does*: “The MVC framework makes it possible for the user to reason about what the system *is*: the *thinking* part of the user cognitive model. But there is little in object orientation, and really nothing in MVC, that helps the developer capture *doing* in the code” ([http://www.artima.com/articles/dci\\_vision.html](http://www.artima.com/articles/dci_vision.html)). DCI attempts to bring the user's cognitive model to the surface in the design of an application.

While *data*, *context*, and *interaction* may be the namesakes of DCI, the arguably most important part of the pattern is the *role*.

*Data* is embodied in the state of objects. Simple objects have hardly any or no logic in them. A *role* contains functionality (read: algorithms) that is abstract. A role comes to life when it is applied to an object, at which point it can execute its algorithms on the data of the object at hand. How to assign specific roles to objects is decided through the *context*. Within a given context, an interaction is an algorithm that performs a use case from the end-user's mental model.

In short, *data* encapsulating the state of a system via *objects*, play a *role* in a particular *context*, which allows for the triggering of a specific *interaction*.

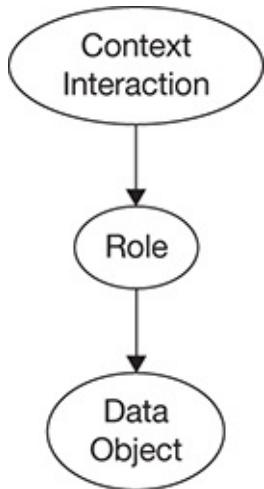
### 6.2.1 DCI and CBRA

How DCI should be implemented in Ruby has been the object of a lot of discussion. One position is that DCI in Ruby is broken (<https://tonyarcieri.com/dci-in-ruby-is-completely-broken>) based on the terrible performance of one possible way of implementing roles. It has been suggested to use SimpleDelegator (<http://www.rubyflow.com/p/6857-dci-that-respects-the-method-cache>), but this leads to issues of self-schizophrenia ([https://www.researchgate.net/profile/Stephan\\_Herrmann/publication/228945903\\_Demystifying\\_object-schizophrenia.pdf](https://www.researchgate.net/profile/Stephan_Herrmann/publication/228945903_Demystifying_object-schizophrenia.pdf)). However, as before, this section will not solve the applicability of DCI in Ruby as a general question. So, what then is the collaboration between DCI and CBRA?

Once again, we must start with an analysis of the relationships between the main constructs of the pattern to understand where CBRA fits in.

DCI creates a partially sorted list of dependencies from simple to complex: Data, as the state of the system embodied in objects, does not depend on anything. Roles, strictly speaking, don't depend on anything either. Roles do, however, have expectations about the methods invokable and fields present on objects. Depending on the language in which DCI is implemented, it may or may not be possible to map this to an explicit dependency, say through the use of interfaces. In effect, data certainly doesn't know about roles; roles know about the data. On top of this is the context in which the interaction of roles is enacted. Both roles and objects are explicitly used within the context. As such, the context must also explicitly depend on roles and objects.

In theory, this means that there are three meaningful components that can be derived from one DCI interaction: context  $\rightarrow$  roles  $\rightarrow$  objects, as depicted in [Figure 6.3](#).



**Figure 6.3.** Dependencies between DCI constructs

Whether splitting a context into components makes sense depends on the larger context of the application. In a complex system, it will most likely be reasonable to split all three parts into separate components. Multiplicity of clients can act as a strong indicator of componentization. If multiple contexts use a role and a role in turn is mixed into multiple objects then all three clearly stand separately. Following this, one would split all three into separate components.

For this section's adaptation of the Sportsball application, we have not introduced a new

component, but have in fact introduced all of the new DCI-based pieces into PredictionUI and Predictor.

## 6.2.2 Implementing DCI with CBRA

This section follows the way Mike Pack's blog post on DCI ([http://mikepackdev.com/blog\\_posts/24-the-right-way-to-code-dci-in-ruby](http://mikepackdev.com/blog_posts/24-the-right-way-to-code-dci-in-ruby)) implements the pattern.<sup>1</sup> As such, it does suffer from the performance issues discussed by Tony Arcieri (<https://tonyarcieri.com/dci-in-ruby-is-completely-broken>).

We are going to analyze how DCI changes the Sportsball application in the area of game prediction. Once again, we are running into the problem of our example's being ridiculously simple. And really the only part of the application that has interesting business logic is the prediction engine. The other parts are CRUD only and wouldn't result in interesting roles, contexts, or interactions.

Let's start with what continues to be the trigger of a prediction, the PredictionsController in PredictorUi.

`./components/prediction_ui/app/controllers/prediction_ui/prec`

[Click here to view code image](#)

```
1 module PredictionUi
2   class PredictionsController < ApplicationController
3     def new
4       @teams = Teams::Team.all
5     end
6
7     def create
8       @prediction = GamePredictionContext.call(
9         Teams::Team.all,
10        Games::Game.all,
11        Teams::Team.find(params["first_team"]["id"]),
12        Teams::Team.find(params["second_team"]["id"]))
13      end
14    end
15  end
16
17
18 class GamePredictionContext
19   def self.call(teams, games, first_team, second_team)
20     GamePredictionContext.new(
21       teams, games, first_team, second_team).call
22   end
23
24   def initialize(teams, games, first_team, second_team)
25     @contenders = teams
26     @contenders.each do |contender|
27       contender.extend Predictor::Role::Contender
28     end
29
30     @hpis = games
31     @hpis.each do |game|
32       game.extend Predictor::Role::HistoricalPerformanceIndicator
```

```

33     end
34
35     @predictor = first_team
36     @predictor.extend Predictor::Role::Contender
37     @predictor.extend Predictor::Role::Predictor
38
39     @second_team = second_team
40     @second_team.extend Predictor::Role::Contender
41   end
42
43   def call
44     @predictor.opponent = @second_team
45     @predictor.contenders = @contenders
46
47     @predictor.learn(@hpis)
48     @predictor.predict
49   end
50 end

```

Line 8 sees the call into the GamePredictionContext, which passes in all the relevant information in one go: teams, games, and the match to predict. Following the controller, currently in the same file, is the definition of the GamePredictionContext itself. As soon as there is a second caller making use of this context, it should surely move into a different file and (most likely) also into its own component.

Within the context, line 19 defines a convenience class method with which we can instantiate and immediately execute the prediction context. The initialize method of the class is where objects get enriched with roles. Every Team object gets the role of a Predictor::Role:: Contender (line 27). Every Game turns into a Predictor::Role:: HistoricalPerformanceIndicator or *hpi* (line 32). Finally, the team passed in via the `first_team` parameter takes the role of the Predictor::Role::Predictor (line 37). All of these are stored as instance variables on the context object.

When the context is executed, which happens via the parameterless `call` method, the second team as well as the contenders are set on `@predictor` (lines 44 and 45). The predictor is trained via the historical performance indicators (i.e., past games). Finally, the prediction can start.

Let us now in turn look at what `Contender`, `HistoricalPerformanceIndicator`, and `Predictor` add to the respective objects.

**`./components/predictor/lib/predictor/role/contender.rb`**

[Click here to view code image](#)

```

1 module Predictor
2   module Role
3     module Contender
4       def self.extended(base)
5         base.rating = [
6           Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
7         ]

```

```

8     end
9
10    def rating
11      @rating
12    end
13
14    def rating=(value)
15      @rating = value
16    end
17
18    def mean_of_rating
19      @rating.first.mean
20    end
21  end
22 end
23 end

```

Contender turns an object into something that is *ratable*. That is, it creates and stores a rating on the object when the object is extended by the module. The module method `self.extended(base)` is called for a module (with the `base` parameter being the extended object) as soon as the object is extended by that module. The `rating` attribute of the object is exposed for reading and writing via `rating` and `rating=` methods. In previous versions of Sportsball, this functionality is contained inline in the Predictor class. In addition, `mean_of_rating` exposes the mean of the rating of a contender in a nice way, which is used during prediction.

`./components/predictor/lib/predictor/role/historical_performance.rb`

[Click here to view code image](#)

```

1 module Predictor
2   module Role
3     module HistoricalPerformanceIndicator
4       def order_of_teams
5         result = [first_team_id, second_team_id]
6         result.reverse! if winning_team != 1
7         result
8       end
9     end
10  end
11 end

```

Besides its epic name, a `HistoricalPerformanceIndicator` does not have much to offer. All it does is allow an object (the game played) to return the teams that played the game in an array with the winning team listed first. This is the data structure which the predictor uses to calculate and update ratings for contenders. Just like Contender, in previous versions of Sportsball, this functionality is also contained inline in the Predictor class.

`./components/predictor/lib/predictor/role/predictor.rb`

[Click here to view code image](#)

```

1 module Predictor
2   module Role
3     module Predictor

```

```

4   def contenders=(contenders)
5     @contenders_lookup =
6       contenders.inject({}) do |memo, contender|
7         memo[contender.id] = contender
8         memo
9     end
10   end
11
12   def opponent=(value)
13     @opponent = value
14   end
15
16   def game_predictable?
17     self != @opponent
18   end
19
20   def learn(games)
21     games.each do |game|
22       game_result = game.order_of_teams.map do |team_id|
23         @contenders_lookup[team_id].rating
24       end
25       Saulabs::TrueSkill::FactorGraph.new(
26         game_result, [1, 2]).update_skills
27       end
28   end
29
30   def predict
31     if game_predictable?
32       ::Predictor::Prediction.new(
33         self,
34         @opponent,
35         likely_winner)
36     else
37       ::Predictor::PredictionError.new(
38         self,
39         @opponent,
40         "Two contenders needed for prediction")
41     end
42   end
43
44   private
45
46   def likely_winner
47     team1 = @contenders_lookup[id]
48     team2 = @contenders_lookup[@opponent.id]
49
50     team1.mean_of_rating > team2.mean_of_rating ? team1 : team2
51   end
52   end
53 end
54 end

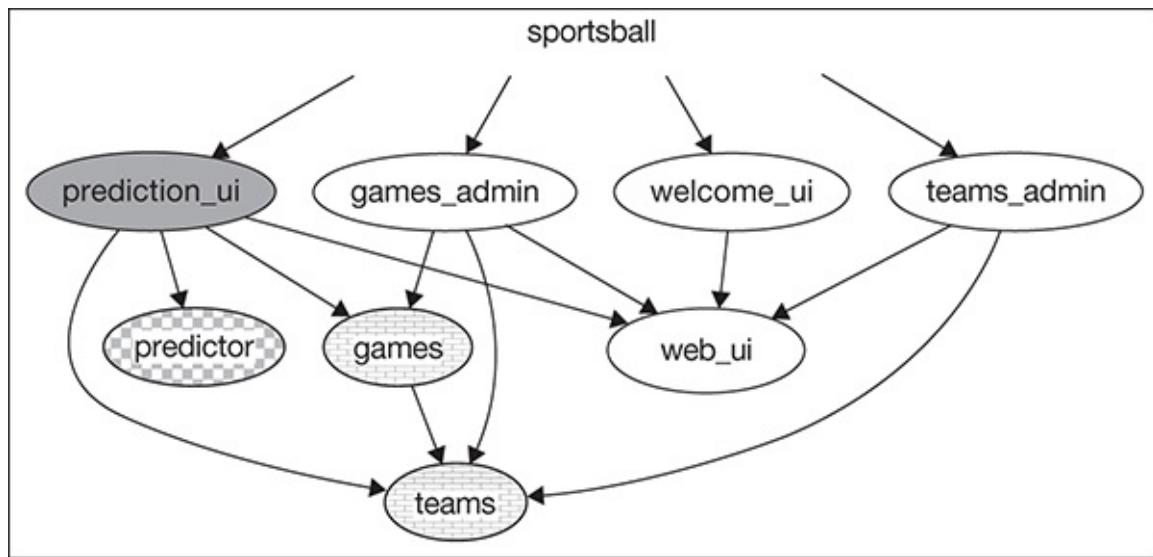
```

The role Predictor is what used to be the Predictor class. Remember that Predictor extends a Team object. Accordingly, where previously we accessed first\_team and second\_team, we now access self and @contender.

The `learn` method (line 20) makes use of the new roles that teams and games have taken as `contenders` and `historical_performance_indicators` in this context. The latter is used to put together the result of a game in line 22. The `Contender` role comes into play when ratings are retrieved in line 23.

During prediction, `likely_winner` (used in line 35) employs the contenders' `mean_of_rating` methods to decide which contender to call as the likely winner of the game.

Coloring the resulting component diagram according to where the constituent parts end up reveals an interesting fact. Let's give context solid fill, role checkboard fill, and object brick-patterned fill. [Figure 6.4](#) shows the result.



**Figure 6.4.** Components in a DCI version of Sportsball

Note that the component containing the role does not explicitly depend on the components containing the data. That is because while all the roles are defined with expectations about the various objects in mind, they are not explicitly tied to them. This is important, as there could be situations in which various objects take on a particular role in different interactions. Actually, the dependency of the role is onto the *interface* of the object. In other languages, where interfaces can be defined explicitly, one might create such an interface, in which case it would live inside of the `predictor` component.

### 6.2.3 Summary

DCI introduces a pattern of object interaction that when first introduced to a codebase will change the way objects are cut into shape and how algorithms and data are connected to each other. It also introduces new, directed dependencies between the pieces that make up the pattern.

As such, DCI introduces a way to think about and organize the codebase. CBRA introduces a way to ensure that that thinking and organizing are first-level citizens and directly visible within the codebase.

In this example, we didn't introduce any new components into the application to switch to a DCI pattern. Whether or not your application will benefit from introducing new

components depends on the complexity and interaction of contexts, roles, and data. The more cases there are in which the same data is mapped to the same or different roles to be used in multiple contexts, the more additional components will make sense to disconnect separate pieces of business functionality.

- 
1. Another, similar blog post is <http://victorsavkin.com/post/13966712168/dci-in-ruby>.

# CHAPTER 7

## Components in Other Languages



Photo: TravelMediaProductions/Shutterstock

The years since I started writing this book have taken me away from Ruby. First the work on Cloud Foundry, then enterprise clients wanting us to work with Java or .NET. I suspect, eventually, we will all do something else. So, how do component-based concepts carry over to other languages? Do they hold up? Is there even such a thing as components?

In short: yes. In fact, as I discuss in the Introduction, other languages have a long history of this kind of stuff. In this last chapter, I want to point you toward some of the ways components are implemented in some other popular languages with which I have come into deeper contact.

The following two examples are from the AppContinuum project I mentioned in the Introduction (<http://www.appcontinuum.io/>). The main reason I chose these is to show just how well component-based application development translates from one language to the next.

## 7.1 Kotlin, Java, and Gradle

Kotlin (<https://kotlinlang.org/>) is a JVM, or Java virtual machine, language. It is being sponsored and developed by JetBrains (the same folks that wrote the IDE, or *integrated development environment*, in which I am writing this). Kotlin is fully compatible with Java and can in fact be mixed into the same projects, some files Java, some files Kotlin. Java needs no introduction.

Kotlin and Java are commonly built using built tools like maven (<https://maven.apache.org/>) or gradle (<https://gradle.org/>). It is through these that we have powerful dependency-management tools at our disposal. I will focus on gradle here, but maven is very similar in principle. With gradle, we specify what our dependencies are and where to find them ([https://docs.gradle.org/current/userguide/artifact\\_dependencies\\_tutorial.html](https://docs.gradle.org/current/userguide/artifact_dependencies_tutorial.html)).

The first example for the AppContinuum is in Kotlin. Get it from <https://github.com/barinek/appcontinuum>. Read the full source code at your own leisure. We are going to focus on the project structure and the component organization piece. The AppContinuum discusses more than just component-based organization, so be sure to git checkout v4—the commit tag in which a component-based architecture is shown.

Check out the directory structure, showing \*.gradle files only. These are the files gradle uses to organize an application.

**Directory structure of the AppContinuum codebase (filtered to \*.gradle files). Execute in the project root (./)**

[Click here to view code image](#)

```
$ tree ./ -P \*.gradle --prune -I test
./
├── app
│   └── build.gradle
└── build.gradle
components
├── accounts
│   └── build.gradle
├── build.gradle
├── jdbc-support
│   └── build.gradle
├── projects
│   └── build.gradle
├── rest-support
│   └── build.gradle
├── test-support
│   └── build.gradle
└── users
    └── build.gradle
        └── settings.gradle
```

If anything about this is surprising, it is how closely it matches the structure we discussed in [Section 3.7](#). ./app contains a Web application while ./components contains several components (often called “projects” or “sub-projects” in Kotlin and Java speak).

The root of the tree contains two gradle files: build.gradle and settings.gradle. These configure the overall artifact that is created and how it is configured.

settings.gradle defines the name of the project we are building (i.e., “appcontinuum”) and specifies the included projects (app and components). It does so in a very clean way.

### **./settings.gradle**

[Click here to view code image](#)

```
1 rootProject.name = "appcontinuum"
2
3 include "app"
4
5 include "components:accounts"
6 include "components:jdbc-support"
7 include "components:projects"
8 include "components:rest-support"
9 include "components:test-support"
10 include "components:users"
```

Without going into detail, build.gradle specifies how to build this app, what plugins to use, the external dependencies, and where to find the source code.

### **./build.gradle**

[Click here to view code image](#)

```
1 buildscript {
2     ext.kotlin_version = '1.0.6'
3     repositories {
4         mavenCentral()
5         jcenter()
6     }
7     dependencies {
8         classpath "org.jetbrains.kotlin:kotlin-gradle-plugin" +
9             ":"$kotlin_version"
10    }
11 }
12
13 subprojects {
14     group "io.barinek.continuum"
15
16     apply plugin: 'kotlin'
17
18     defaultTasks "clean", "build"
19
20     repositories {
21         mavenCentral()
22         jcenter()
23     }
24
25     dependencies {
26         compile "org.jetbrains.kotlin:kotlin-stdlib:" +
27             "$kotlin_version"
28         compile "org.jetbrains.kotlinx:kotlinx-support-jdk8:" +
```

```

29         "0.3"
30     compile 'com.fasterxml.jackson.core:' +
31             'jackson-core:2.8.4'
32     compile 'com.fasterxml.jackson.core:' +
33             'jackson-databind:2.8.4'
34     compile 'com.fasterxml.jackson.core:' +
35             'jackson-annotations:2.8.4'
36
37     compile "com.fasterxml.jackson.datatype:" +
38             "jackson-datatype-jsr310:2.8.4"
39     compile "com.fasterxml.jackson.module:" +
40             "jackson-module-kotlin:2.8.4"
41
42     compile group: 'org.slf4j', name: 'slf4j-api',
43             version: '1.7.10'
44     compile group: 'org.slf4j', name: 'slf4j-simple',
45             version: '1.7.10'
46
47     testCompile 'junit:junit:4.11'
48     testCompile "org.jetbrains.kotlin:kotlin-test-junit:" +
49             "$kotlin_version"
50 }
51
52 sourceSets {
53     main.kotlin.srcDir "src/main/kotlin"
54     test.kotlin.srcDir "src/test/kotlin"
55
56     test.resources.srcDir "src/test/resources"
57 }
58 }
```

To give just one example of a build.gradle file in one of the components, let's look at accounts. We see here that it is dependent on several of the other components from this app, namely, jdbc-support, rest-support, test-support, and users.

**./components/accounts/build.gradle**

[Click here to view code image](#)

```

1 version "1.0-SNAPSHOT"
2
3 dependencies {
4     compile project(":components:jdbc-support")
5     compile project(":components:rest-support")
6     compile project(":components:test-support")
7     compile project(":components:users")
8 }
```

## 7.2 .NET / C

Mike Barinek was nice enough to also publish a version of the AppContinuum in .NET: [https://github.com/barinek/appcontinuum\\_dotnet](https://github.com/barinek/appcontinuum_dotnet).

Here is its directory structure. For .NET, the component structure is organized via C# project files.

## Directory structure of the AppContinuum .NET codebase (filtered to \*.csproj files). Execute in the project root (./)

[Click here to view code image](#)

```
$ tree ./ -P *.csproj --prune
./
├── Applications
│   ├── AllocationsServer
│   │   └── AllocationsServer.csproj
│   ├── BacklogServer
│   │   └── BacklogServer.csproj
│   ├── RegistrationServer
│   │   └── RegistrationServer.csproj
│   └── TimesheetsServer
│       └── TimesheetsServer.csproj
└── Components
    ├── Accounts
    │   └── Accounts.csproj
    ├── AccountsTest
    │   └── AccountsTest.csproj
    ├── Allocations
    │   └── Allocations.csproj
    ├── AllocationsTest
    │   └── AllocationsTest.csproj
    ├── Backlog
    │   └── Backlog.csproj
    ├── BacklogTest
    │   └── BacklogTest.csproj
    ├── DatabaseSupport
    │   └── DatabaseSupport.csproj
    ├── DatabaseSupportTest
    │   └── DatabaseSupportTest.csproj
    ├── Projects
    │   └── Projects.csproj
    ├── ProjectsTest
    │   └── ProjectsTest.csproj
    ├── TestSupport
    │   └── TestSupport.csproj
    ├── Timesheets
    │   └── Timesheets.csproj
    ├── TimesheetsTest
    │   └── TimesheetsTest.csproj
    └── Users
        └── Users.csproj
    └── UsersTest
        └── UsersTest.csproj
```

To get a good comparison with the previous example in gradle, let's look further at the project file for the Accounts component. This is less clean than what we saw earlier—it is XML, after all. However, if you look for the elements named ProjectReference, you can once again see the dependencies: DatabaseSupport, Users, and MySqlConnector.

**./components/accounts/build.gradle**

[Click here to view code image](#)

```
1 <Project Sdk="Microsoft.NET.Sdk">
2
3   <PropertyGroup>
4     <TargetFramework>netstandard1.6</TargetFramework>
5   </PropertyGroup>
6
7   <ItemGroup>
8     <ProjectReference
9       Include="..../DatabaseSupport/DatabaseSupport.csproj" />
10    <ProjectReference Include="..../Users/Users.csproj" />
11    <PackageReference Include=" MySqlConnector" Version="0.19.3" />
12  </ItemGroup>
13
14  <ItemGroup>
15    <PackageReference Include=" Microsoft.AspNetCore"
16      Version="1.1.2" />
17    <PackageReference Include=" Microsoft.AspNetCore.Mvc"
18      Version="1.1.3" />
19    <PackageReference Include=" Microsoft.Extensions.Logging.Debug"
20      Version="1.1.2" />
21  </ItemGroup>
22
23 </Project>
```

## 7.3 Conclusion

The key takeaway of the preceding is simply this.

There is always a lot to learn when getting into new environments, especially when starting development in new programming languages and frameworks. This will take dedication and attention to detail. Note, however, how I introduced no Java, no Kotlin, no C#. We looked not for language syntax but for application structure. As you are learning about new technologies, be on the lookout for the larger patterns you are creating with every small code change. These larger patterns don't change as quickly, and they do not differ that much among all those different systems.

Ask yourself in every context whether you are still able to tell where something belongs—and where it does not belong. And ask whether you can still tell who else and what other work you are dependent upon. If you no longer know, step back, take a break, and see if you can create something that will help you answer these questions. Create a structure that will continually challenge you to write the best code you can today. It will make your code better tomorrow.

---

# Appendix

The following contains some additional discussions on engine-specific aspects of component-based applications.

## A.1 Plain versus --full versus --mountable Engines

As we saw in [Section 2.1](#), there is a decision to make when it comes to the command-line parameters used for the creation of an engine. Do we want to create a *plain plugin*, a *full engine*, a *mountable engine*, or maybe even *just a gem (no plugin)*? There are quite a few command line parameters in addition to this, but we will ignore most of them for now and concentrate on the different kinds of plugins we can create.

[Click here to view code image](#)

```
$ bundle gem bundle_gem  
$ rails plugin new plain_plugin  
$ rails plugin new full_engine --full  
$ rails plugin new mountable_engine --mountable  
$ rails plugin new full_mountable_engine --full --mountable
```

---

### Which Type of Plugin Should You Use?

In most cases you will want to use mountable engines for CBRA apps. They are as isolated as possible from the main app. Also, once we get to different types of components, we will see that mountable engines can easily be morphed into any other type of component we might want to create simply by deleting some folders and files. Most of the engines that are extracted in [Chapter 4](#) are of this type.

Any engine, any type of plugin, will depend on Rails or some part of it. In situations where you do not need or do not want a dependency on Rails you should start with a straight-up gem. The Predictor gem that is extracted in [Section 4.2](#) falls in this category, as well as several of the gems that are created in conjunction with architecture patterns as discussed in [Chapter 6](#).

---

Let's look at each of these options in turn.

## A.1.1 Gem

To create a gem without reference to Rails, we can use `bundle gem` or, because there is really not much to it, make it from scratch (<http://guides.rubygems.org/make-your-own-gem/>).

### Directory structure of a gem generated with `bundle gem`

[Click here to view code image](#)

```
bundle_gem
└── bin
└── lib
    └── bundle_gem
└── spec
```

`bundle gem` creates a git configuration for your new gem. In the context of a component-based application, this is typically not desirable, as we already have a git repository for the container Rails application. We remove all traces of git from our gem quickly via `rm -rf bundle_gem/.git*`. We need to remove the `.git` directory for a gem so that its code is checked in correctly in the application's git root. That said, it is not strictly necessary to remove the `.gitignore` file. One could instead leave these per engine, as git supports ignoring files per directory. I have found in practice that this doesn't add much value (but a lot of `.gitignore` files), so I still tend to remove all git-related files from components.

In gems, code implementing the domain commonly is added within the namespace of the gem. For the preceding gem, that means that files are added under `bundle_gem/lib/bundle_gem`.

## A.1.2 Plain Plugin

`rails plugin new` without any further parameters creates a plain plugin. In all my experience with CBRA, I have not found a use case for plain plugins.

### Directory structure of a plain plugin

[Click here to view code image](#)

```
plain_plugin
└── bin
└── lib
    └── plain_plugin
        └── tasks
└── test
└── dummy
    └── app
        └── assets
            ├── config
            ├── images
            ├── javascripts
            │   └── channels
            │   └── stylesheets
            └── channels
                └── application_cable
```

```

    └── controllers
        └── concerns
    └── helpers
    └── jobs
    └── mailers
    └── models
        └── concerns
    └── views

└── layouts
    └── bin
    └── config
        └── environments
            └── initializers
        └── locales
    └── db
    └── lib
        └── assets
    └── log
    └── public
        └── tmp
            └── cache
            └── assets

```

The first thing to note about the directory structure is that there is hardly any content in `lib`, but a lot inside of `test`. This is because a plain plugin does not generate a `Rails::Engine` and also does not create its supporting directory structure. Instead, it generates the shell of an ordinary gem, as discussed. The fact that there is a `tasks` folder is a cosmetic difference; there is only a placeholder file in there.

A real difference is that the gem we generated has a dependency on the `rails` gem as specified in the `gemspec`: `s.add_dependency "rails", "~> 5.1.0.rc2"`. Notice that this dependency is not just a *development dependency*, but a *runtime dependency*.

However, the biggest difference is the `test/dummy` folder. It contains the so-called *dummy Rails app*, a reduced Rails app that is used to load the engine in the context of tests. The following two pieces of code are excerpts from files in the dummy app. The code from `application.rb` shows how the gem is loaded into the dummy app. In `test_helper.rb`, we see that it is the dummy app that is loaded in order to load application code for tests.

## **Top of `./test/dummy/config/application.rb`**

[Click here to view code image](#)

```

1 require_relative 'boot'
2
3 require 'rails/all'
4
5 Bundler.require(*Rails.groups)
6 require "plain_plugin"

```

## **Top of `./test/test_helper.rb`**

[Click here to view code image](#)

```
1 require File.expand_path("../test/dummy/config/environment.rb",
2     __FILE__ )
3 ActiveRecord::Migrator.migrations_paths = [
4     File.expand_path("../test/dummy/db/migrate", __FILE__ )
5 ]
6 require "rails/test_help"
7
8 # Filter out Minitest backtrace while allowing backtrace
9 # from other libraries to be shown.
10 Minitest.backtrace_filter = Minitest::BacktraceFilter.new
```

This dummy app is a common feature of all plugins and we will see it again in the folder structures of full and mountable engines.

---

## What to Do About the Dummy App If You Use RSpec?

`rails plugin new my_rspec_component --dummy-path=spec/dummy --skip-test-unit` will create the dummy app under spec and it will prevent `test_helper.rb` from being created. Now you can add rspec as a dependency to the gemspec of your gem and use `rspec --init` to create `spec_helper.rb`. Paste in the lines shown from the top of `test_helper.rb` to make RSpec work with the dummy app setup and load the gem's files through it.

---

To summarize, creating a plain plugin creates a gem that has `rails` as a runtime dependency and that features the dummy app concept for testing the gem in the context of a Rails app. I have never found a use for plain plugins in the context of CBRA. I either seem to want more, namely a `Rails::Engine` (as we will see in the next two sections), or I want less, namely a gem that does not have a dependency on `rails` at all. `Rails` is such a large dependency itself that I highly recommend not adding it to anything that does not absolutely need it.

---

## Rails as a Dependency

Rails is a *massive* dependency. Do not add it to anything that does not absolutely need it!

---

### A.1.3 Full Engine

`rails plugin new full_engine --full` creates a full engine. Full engines are a big departure from plain plugins. I commonly use full engines for one specific purpose: to get started with CBRA in an existing codebase. [Chapter 5](#) covers this use in depth.

Full engines have the following directory structure.

#### Directory structure of a full engine (contents of `test` folder omitted)

[Click here to view code image](#)

```
full_engine
```

```
└── app
    ├── assets
    │   ├── config
    │   ├── images
    │   │   └── full_engine
    │   ├── javascripts
    │   │   └── full_engine
    │   └── stylesheets
    │       └── full_engine
    ├── controllers
    ├── helpers
    ├── mailers
    ├── models
    └── views
    └── bin
    └── config
    └── lib
        └── full_engine
            └── tasks
    └── test
        └── dummy
```

Here, you see the big difference between engine and non-engine gems: Gems that create engines have an `app` folder, which has exactly the subdirectories that you would expect from any Rails app's `app` folder.

We also get a `bin` folder, containing a `rails` executable (in some version of Rails this is `script/rails`), which allows us to run `rails` commands in the root of the gem. You can use it to have models, controllers, or scaffolds generated within the engine. Note that running `rails s` will not work. It will fail with the following error message:

#### [Click here to view code image](#)

```
Error: Command not recognized
Usage: rails COMMAND [ARGS]
```

```
The common Rails commands available for engines are:
generate Generate new code (short-cut alias: "g")
destroy Undo code generated with "generate" (short-cut alias: "d")
```

All commands can be run **with -h for** more information.

If you want to run any commands that need to be run **in** context of the application, like `'rails server'` or `'rails console'`, you should **do** it from the application's directory (typically `test/dummy`).

The last lines in this output point us to the problem. The `rails` command we execute from the root directory of the engine is run in the context of the *engine* and not in that of a Rails *application*. And Rails engines have no servers. To run the “engine’s server,” we have to call `rails` from the `spec/dummy` directory.

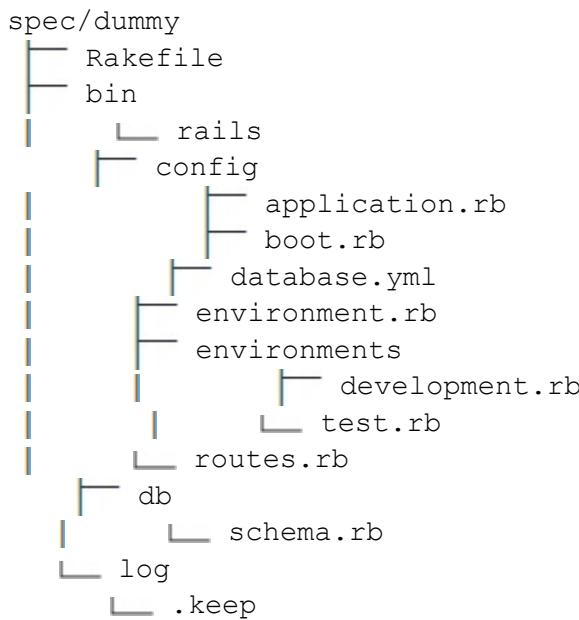
---

## “The Dummy App Contains So Much Code—I Don’t Want It!”

The dummy app adds a significant amount of code that needs to be maintained.

Fortunately, in practice, there is not a lot that ever changes. The biggest driver for change in the dummy app is an upgrade of Rails versions—although even that often does not require any changes. Nonetheless, it is possible to reduce the footprint of the dummy app by removing all files except for these listed:

[Click here to view code image](#)



Be careful when removing initializers from the dummy app; ideally, leave those that are also used in the main application in the dummy app so that tests running within the dummy app run against a similarly configured environment.

---

The file defining the `Rails::Engine` is in `./lib/full_engine/engine.rb` and looks as follows:

**`./lib/full_engine/engine.rb`**

[Click here to view code image](#)

```
1 module FullEngine
2   class Engine < ::Rails::Engine
3   end
4 end
```

There is now a `routes.rb` file in the `config` folder. While it does not yet have any routes specified, it is interesting to note its contents. The first line, `Rails.application.routes.draw do`, is exactly what you find at the top of a Rails app's `routes.rb` (and indeed, what you will find in the `routes` file of the dummy app).

**`./config/routes.rb`**

[Click here to view code image](#)

```
1 Rails.application.routes.draw do
2 end
```

I tend to not use full engines inside of CBRA applications because they do not achieve the higher level of separation from the main app (and also from other engines) that I find so valuable. This is because all app classes (models, controller, helpers, etc.) are added to

the root module (just like in every Rails app). That means if we run `rails g scaffold user name:string` in the engine, it is going to create a class `User < ActiveRecord::Base` just like it would in the Rails app. This class would conflict with a `User` class defined in a different engine and the final state of `User` would depend on their load order. As we saw before, the engine's routes are also drawn directly into the Rails app's routes. Again, you can ask what would happen if the app defined a route of the same name. And again, the answer is that it depends on the load order.

In practice, the gem is always going to be loaded before the application code and so the behavior of these conflicts will be deterministic. However, if we could prevent these conflicts from ever happening, we would be in a much better situation. And it turns out that mountable engines do just that.

#### A.1.4 Mountable Engine

Mountable engines are the type of engine I suggest you use every time you would like to componentize part of your Rails app and need access to Rails' functionality.

These engines are created by `rails plugin new mountable_engine --mountable`. Mountable engines are always full engines as well, and whether you add the parameter `--full` or not has no effect on what is created. The generated directory structure (again omitting the dummy app in `test`) is as follows:

#### Directory structure of a mountable engine (contents of `test` folder omitted)

[Click here to view code image](#)

```
mountable_engine
├── app
│   ├── assets
│   │   ├── config
│   │   ├── images
│   │   │   └── mountable_engine
│   │   ├── javascripts
│   │   │   └── mountable_engine
│   │   └── stylesheets
│   │       └── mountable_engine
│   ├── controllers
│   │   └── mountable_engine
│   ├── helpers
│   │   └── mountable_engine
│   ├── jobs
│   │   └── mountable_engine
│   ├── mailers
│   │   └── mountable_engine
│   ├── models
│   │   └── mountable_engine
│   └── views
│       ├── layouts
│       │   └── mountable_engine
└── bin
└── config
└── lib
```

```
|   └── mountable_engine
|       └── tasks
└── test
    └── dummy
```

Note that most of the folders under app now have a subfolder `mountable_engine`. This is because mountable engines are created in an *isolated namespace*. Let's look at the file defining the Rails engine to explain what this means:

**`./lib/mountable_engine/engine.rb`**

[Click here to view code image](#)

```
1 module MountableEngine
2   class Engine < ::Rails::Engine
3     isolate_namespace MountableEngine
4   end
5 end
```

Line 3, `isolate_namespace MountableEngine`, turns the entire engine into one that uses the module name `MountableEngine` for every class it creates. This means that the example mentioned before, namely `rails g model user name:string`, would no longer create a conflict between the engine and the app because in the engine we would get the following file `user.rb` in `app/models/mountable_engine/`:

**`./lib/mountable_engine/app/models/mountable_engine/user.rb`**

[Click here to view code image](#)

```
1 module MountableEngine
2   class User < ApplicationRecord
3   end
4 end
```

This is a big deal; we now have a built-in mechanism to isolate the code between our components as well as is possible with Ruby. Of course, you could manually create a `user.rb` directly in `app/model` and not namespace it. Nothing in Ruby is truly private, and everything in Ruby can be reopened and messed with. But setting the right defaults and writing exemplary code is a huge step in the right direction (and I would argue that it is also a *big enough* step).

There is another change in comparison to the full engine: its routes. They are no longer mounted directly into the Rails app but instead defined as part of the engine. As we already saw in [Section 2.1](#), this means that we first need to mount the engine into the main Rails app for its routes to be loaded—hence its name. We will take a closer look at how this works in the next section.

## A.2 How Do Engine Routes and Engine Mounting Work?

In the previous section, we saw that a mountable engine can hold on to its own routes. Here is the full `routes.rb` from the mountable engine created in the previous section.

**`./lib/mountable_engine/config/routes.rb`**

[Click here to view code image](#)

```
1 MountableEngine::Engine.routes.draw do
2   resources :users
3 end
```

We can confirm that there are no engine routes in the host Rails app by checking its routes.

**./test/dummy/config/routes.rb**

[Click here to view code image](#)

```
1 Rails.application.routes.draw do
2   mount MountableEngine::Engine => "/mountable_engine"
3 end
```

The dummy app mounts the engine under the underscored version of the engine's name by default. We can confirm this by checking the output of `rake routes`. However, as we are dealing with a dummy app, we need to call `rake app:routes` instead.

**Output of `rake app:routes*`**

[Click here to view code image](#)

Prefix	Verb	URI	Pattern	Controller#Action
mountable_engine			/mountable_engine	MountableEngine::Engin

Routes for MountableEngine::Engine:

users	GET	/users(.:format)	moun...ngine/users#index
	POST	/users(.:format)	moun...ngine/users#create
new_user	GET	/users/new(.:format)	moun...ngine/users#new
edit_user	GET	/users/:id/edit(.:format)	moun...ngine/users#edit
user	GET	/users/:id(.:format)	moun...ngine/users#show
	PATCH	/users/:id(.:format)	moun...ngine/users#update
	PUT	/users/:id(.:format)	moun...ngine/users#update
	DELETE	/users/:id(.:format)	moun...ngine/users#destroy

As we can see, the routes formatter prints out which routes are created by the main app (or dummy app, as in this case) and which come from an engine. Similar to the argument for the isolation of classes, we can say that with mountable engines, defaults have been set to isolation: There will be no conflict between the routes defined inside of an engine and those defined on the outside as long as we ensure that the engine has a unique mountpoint.

---

## Accessing Dummy App Rake Tasks

`rake -T` reveals an oddity about the way `rake` tasks are loaded inside of an engine. We see all the standard Rails `rake` tasks, but there is an `app:` prefix in front of all of them. And some seem to be duplicated.

**Partial output of `rake -T` (showing `app:` prefix)**

[Click here to view code image](#)

```
rake app:about
  # List versions of all Rails frame
  works and the environment
rake app:app:template
  # Applies the template supplied by LOCATION=(/path/to/template)
  # or URL
rake app:app:update
```

```
# Update configs and some other initially generated files
# (or use just update:configs or update:bin)
rake app:assets:clean[keep]
  # Remove old compiled assets
rake app:assets:clobber
  # Remove compiled assets
rake app:assets:environment
  # Load asset compile environment
rake app:assets:precompile
  # Compile all the assets named in config.assets.precompile
rake app:cache_digests:dependencies
  # Lookup first-level dependencies for TEMPLATE
  # (like messages/show or comments/_comment.html)
rake app:cache_digests:nested_dependencies
  # Lookup nested dependencies for TEMPLATE
  # (like messages/show or comments/_comment.html)
rake app:db:create
  # Creates the database from DATABASE_URL or config/ database.yml
  # for the current RAILS_ENV (use db:create:all to create all
  # databases in the config). Without RAILS_ENV or when RAILS_ENV
  # is development, it defaults to creating the development and
  # test databases
rake app:db:drop
  # Drops the database from DATABASE_URL or config/ database.yml
  # for the current RAILS_ENV (use db:drop:all to drop all databases
  # in the config). Without RAILS_ENV or when RAILS_ENV is
  # development, it defaults to dropping the development and
  # test databases
rake app:db:environment:set
  # Set the environment value for the database
rake app:db:fixtures:load
  # Loads fixtures into the current environment's database
rake app:db:migrate
  # Migrate the database
  # (options: VERSION=x, VERBOSE=false, SCOPE=blog)
rake app:db:migrate:status
  # Display status of migrations
rake app:db:rollback
  # Rolls the schema back to the previous version
  # (specify steps w/ STEP=n)
rake app:db:schema:cache:clear
  # Clears a db/schema_cache.yml file
rake app:db:schema:cache:dump
  # Creates a db/schema_cache.yml file
rake app:db:schema:dump
  # Creates a db/schema.rb file that is portable against
  # any DB supported by Active Record
rake app:db:schema:load
  # Loads a schema.rb file into the database
rake app:db:seed
  # Loads the seed data from db/seeds.rb
rake app:db:setup
  # Creates the database, loads the schema, and initializes
  # with the seed data (use db:reset to also drop the database first)
rake app:db:structure:dump
  # Dumps the database structure to db/structure.sql
```

```

rake app:db:structure:load
  # Recreates the databases from the structure.sql file
rake app:db:version
  # Retrieves the current schema version number
rake app:dev:cache
  # Toggle development mode caching on/off
rake app:initializers
  # Print out all defined initializers in the order they are
  # invoked by Rails
rake app:log:clear
  # Truncates all/specify *.log files in log/ to zero bytes
  # (specify which logs with LOGS=test,development)
rake app:middleware
  # Prints out your Rack middleware stack
rake app:mountable_engine:install:migrations
  # Copy migrations from mountable_engine to application
rake app:notes
  # Enumerate all annotations
  # (use notes:optimize, :fixme, :todo for focus)
rake app:notes:custom
  # Enumerate a custom annotation, specify with ANNOTATION=CUSTOM
rake app:restart
  # Restart app by touching tmp/restart.txt
rake app:routes
  # Print out all defined routes in match order, with names
rake app:secret
  # Generate a cryptographically secure secret key (this is
  # typically used to generate a secret for cookie sessions)
rake app:test
  # Runs all tests in test folder except system ones
rake app:test:db
  # Run tests quickly, but also reset db
rake app:test:system
  # Run system tests only
rake app:time:zones[country_or_offset]
  # List all time zones, list by two-letter country code
  # (`rails time:zones[US]`), or list by UTC offset (`rails
  # time:zones[-8]`)
rake app:tmp:clear
  # Clear cache and socket files from tmp/
  # (narrow w/ tmp:cache:clear, tmp:sockets:clear)
rake app:tmp:create
  # Creates tmp directories for cache, sockets, and pids
rake app:update
  # Update some initially generated files
rake app:yarn:install
  # Install all JavaScript dependencies as specified via Yarn

```

## Partial output of `rake -T` (showing engine-provided tasks)

[Click here to view code image](#)

```

rake build
  # Build mountable_engine-0.1.0.gem into the pkg directory
rake clean
  # Remove any temporary products
rake clobber

```

```

# Remove any generated files
rake clobber_rdoc
# Remove RDoc HTML files
rake db:create
# Create the database from config/database.yml for the current
# Rails.env
# (use db:create:all to create all databases in the config)
rake db:drop
# Drops the database for the current Rails.env
# (use db:drop:all to drop all databases)
rake db:fixtures:load
# Load fixtures into the current environment's database
rake db:migrate
# Migrate the database (options: VERSION=x, VERBOSE=false)
rake db:migrate:status
# Display status of migrations
rake db:rollback
# Rolls the schema back to the previous version
# (specify steps w/ STEP=n)
rake db:schema:dump
# Create a db/schema.rb file that can be portably
# used against any DB supported by Active Record
rake db:schema:load
# Load a schema.rb file into the database
rake db:seed
# Load the seed data from db/seeds.rb
rake db:setup
# Create the database, load the schema, and initialize
# with the seed data (use db:reset to also drop the db first)
rake db:structure:dump
# Dump the database structure to an SQL file
rake db:version
# Retrieves the current schema version number
rake install
# Build and install mountable_engine-0.1.0.gem into system gems
rake install:local
# Build and install mountable_engine-0.1.0.gem into system gems
# without network access
rake rdoc
# Build RDoc HTML files
rake release[remote]
# Create tag v0.1.0 and build and push mountable_engine-0.1.0.gem      # to
Rubygems
rake rerdoc
# Rebuild RDoc HTML files
rake stats
# Report code statistics (KLOCs, etc) from the application or engine
rake test
# Run tests

```

The tasks with the `app:` prefix are the tasks that we have access to because the dummy app's `rake` tasks are loaded into the engine. The `rake` tasks that come directly from the engine (user-defined or not) come directly from the engine defined in this gem.

---

Unique mountpoints are easy to verify in `routes.rb` of the main app. In case there are multiple engines, they should all be mounted to a unique path. If there are any other routes defined in the main app, ensure that none of them produce path prefixes that are the same as the prefix of an engine.<sup>1</sup>

This is why we were able to mount our first engine at `/`, the root, in [Section 2.1](#): The main app did not create any routes and there was only one engine. Trivially, there are no path collisions in that case.

### A.2.1 Engines as Subdomains

Courtesy of Sean Collins comes an alternative to using distinct subpaths for each engine, namely, using different subdomains for each engine. To achieve this, `config/routes.rb` needs to change:

#### `config/routes.rb`

[Click here to view code image](#)

```
1 Rails.application.routes.draw do
2   constraints subdomain: "sportsball" do
3     mount AppComponent::Engine, at: "/"
4   end
5 end
```

In order for this to work on the dev machine (assuming OSX), we have to add corresponding lines to the `/etc/hosts` file:

#### `Required entries in /etc/hosts`

[Click here to view code image](#)

```
1 # For rails dev, so we can use subdomains
2 127.0.0.1 dev.localhost
3 127.0.0.1 sportsball.dev.localhost
```

Now, we can go to `sportsball.dev.localhost:3000`, and it works!

Note that you cannot make your domains end in `.dev`, like `local.dev`, because `.dev` is valid generic top-level domain (<https://iyware.com/dont-use-dev-for-development/>).

There are a bunch of domains out there that can potentially make this approach even simpler. `localtest.me`, `lvh.me`, and `vcap.me` all have their DNS set to create a loopback wildcard URL with `127.0.0.1` as the target. Effectively, this means that you can simply find your subdomains at `dev.localtest.me` and `sportsball.localtest.me` without any change to your hosts file. All you have to do is to ensure you bind your Rails app to the respective domain on startup, like so: `rails s -b localtest.me`.

Scott Forsyth runs `localtest.me` and discusses security implications (<https://weblogs.asp.net/owscott/introducing-testing-domain-localtest-me>). If you are worried about potential attack vectors, read further info there and consider running your own loopback server. Levi Cook, owner of `lvh.me`, discusses SSL for this method (<https://gist.github.com/levicook/563675>).

## A.2.2 Engines Are (Unfortunately) Singletons

I have seen recurring discussions for a particular use case of engines: What if we could mount an engine multiple times to provide functionality in different contexts? For example, what if an admin engine could be mounted within different engines in which it would then provide contextualized administration functions (<https://groups.google.com/d/msg/components-in-rails/T7mK2sktGi0/xbw0B9ktAUJ>)?

Unfortunately, this does not really work.

When mounting engines, note that we mount them using the engine's classname, like `mount MountableEngine::Engine`. While it is possible to mount an engine multiple times, like so,

[Click here to view code image](#)

```
1  mount Admin::Engine, at: "admin1", as: "admin1"
2  mount Admin::Engine, at: "admin2", as: "admin2"
```

the sample application, `cbra_multi_mount` ([https://github.com/shageman/cbra\\_multi\\_mount](https://github.com/shageman/cbra_multi_mount)), running at <https://cbra-multi-mount.herokuapp.com/>, shows the problem. The mountpoint registered last for the engine wins and all path helpers, no matter in which engine's context they are called, will return the same paths. At the same time, both URLs `cbra-multi-mount.herokuapp.com/admin1/things` and `cbra-multi-mount.herokuapp.com/admin2/things` work.

As long as engine multi-mounting is not a thing built natively into Rails, I recommend avoiding seeking solutions like this.

## A.3 Additional Testing with Older Versions of Rails

In versions of Rails before 5, view and routing tests were independent from controller tests. With small adaptations, they can be made to work for component-based applications.

### A.3.1 View Specs

View specs for engines look very much like view specs for Rails apps. Here is a brief example (we will skip the implementation for this one):

```
./components/app_component/spec/views/app_component/app/teams
new.html.slim_spec.rb
```

[Click here to view code image](#)

```
1 RSpec.describe "app_component/teams/new", :type => :view do
2   before :each do
3     assign :team, AppComponent::Team.new(
4       :name => "MyString"
5     )
6   end
7
8   it "renders new team form" do
9     render
```

```

10     assert_select "form[action=?][method=?]", 
11         app_component.teams_path, "post" do
12     assert_select "input#team_name[name=?]", "team[name]"
13   end
14 end
15 end

```

Once again, we need a hack to get these specs to work, as the engine's routes are not loaded by default. This time, however, the problem does not lie with the spec, but with the views themselves (more specifically, how they are loaded into the tests by RSpec). Within the views, paths cannot be loaded without another hack. However, if you never use named routes in your views, you will not see any problems. If you do see problems, another file needs to be created in spec/support:

**./components/app\_component/spec/support/view\_spec\_fixes.rb**

[Click here to view code image](#)

```

1 ActionView::TestCase::TestController.instance_eval do
2   helper AppComponent::Engine.routes.url_helpers
3 end
4 ActionView::TestCase::TestController.class_eval do
5   def _routes
6     AppComponent::Engine.routes
7   end
8 end

```

## A.3.2 Routing Specs

Just like controller specs, routing specs use the `routes` method to ensure that the engine's routes are available for tests, as can be seen in line 2 in the following:

**./components/app\_component/spec/routing/app\_component/teams\_r**

[Click here to view code image](#)

```

1 RSpec.describe AppComponent::TeamsController, :type => :routing do
2   routes { AppComponent::Engine.routes }
3
4   it "routes to #index" do
5     expect(:get => "/teams").to
6       route_to("app_component/teams#index")
7   end
8
9   it "routes to #new" do
10    expect(:get => "/teams/new").to route_to(
11      "app_component/teams#new")
12  end
13
14  it "routes to #show" do
15    expect(:get => "/teams/1").to
16      route_to("app_component/teams#show", id: "1")
17  end
18
19  it "routes to #edit" do
20    expect(:get => "/teams/1/edit").to
21      route_to("app_component/teams#edit", id: "1")

```

```

22 end
23
24 it "routes to #create" do
25   expect(:post => "/teams").to
26     route_to("app_component/teams#create")
27 end
28
29 it "routes to #update" do
30   expect(:put => "/teams/1").to
31     route_to("app_component/teams#update", id: "1")
32 end
33
34 it "routes to #destroy" do
35   expect(:delete => "/teams/1").to
36     route_to("app_component/teams#destroy", id: "1")
37 end
38 end

```

Other than that, routing specs only need to be adapted with regard to the mappings that are to be identified. For the preceding example, the TeamsController, all routes are routed to app\_component/teams, which captures the fact that the controller is within the AppComponent namespace.

# Index

---

## A

ActiveAdmin gem, 29  
ActiveRecord model specs, 51–54  
API components, 190–191  
application continuum, 14–15  
applications, refactoring components from. *See* refactoring components  
asset loading, 74–82

## B

bottom-up approach to component extraction  
explained, 112–120  
Predictor component extraction, 120–139

## C

C#, 241–243  
capybara gem, 64  
CarrierWave uploader, 201–202, 204  
cbratools gem, 188  
CI (continuous integration) server setup  
Jenkins, 70–72  
Travis CI, 72–74, 102–103  
classes, public interface of, 51  
Cloud Foundry, 86, 87–90, 103–109  
cobradeps gem, 4, 26, 138

collaboration, 6–7

common functionality components, 192–193

Commentator, 29

communication of intent, 4–5

complexity, 10–12

component diagrams, 3–4, 26–27

component initializers, 80

component-based applications

in application continuum, 14–15

benefits of, 4–13

creating, 18–27

deploying to PaaS, 86–90, 103–109

component-based Ruby, 13

components. *See also* refactoring components

API components, 190–191

asset loading, 74–82

common functionality components, 192–193

dependencies

locking down gem versions, 38–40

with path blocks, 34–35

slim-rails gem usage, 35–37

trueskill gem usage, 40–43

updating, 90–99

extracting

bottom-up approach, 112–120

Predictor component, 120–139

pulling up UI components, 150–167

pushing down model components, 168–183

top-down approach, 139–150

generating, 21, 151

in Kotlin, Java, gradle, 238–241

migrations within, 27–34

naming, 20–21  
in .NET, C#, 241–243  
objects versus, 1–2  
renaming, 183–188  
splitting, 190  
testing  
    ActiveRecord model specs, 51–54  
    controller specs, 59–62  
    development dependencies, 50–51  
    feature specs, 64–65  
    helper specs, 62–64  
    model components, 170–183  
    non-ActiveRecord model specs, 54–59  
    third-party service adapters, 191–192  
comprehension of application parts, 9–12  
context (in DCI), 227  
controllers  
    API components, 190–191  
    generating, 24  
    in Hexagonal Architecture, 215–219  
    testing, 59–62

## D

data (in DCI), 227  
databases  
    in Hexagonal Architecture, 219–226  
    migrations, 89–90  
    renaming tables, 187  
    switching, 82–85  
DCI (data-context-integration), 227–235  
    dependencies, 228  
PredictionController class, 229–231

Predictor role, 233–234

ratable objects, 231–232

deleting

app folder, 18–19

files and directories, 208

gem files, 121

dependencies

in component refactoring, 122

AppComponent dependencies, 132–137

lack of, for Predictor component, 142

mapping, 140–141

model dependencies, 170–183

persistence component, 202–205

removing, 128–131

UI dependencies, 142–150

development dependencies, 50–65

runtime dependencies

locking down gem versions, 38–40

objects versus components, 1–2

with path blocks, 34–35

sample dependency graph, 5

slim-rails gem usage, 35–37

trueskill gem usage, 40–43

updating, 90–99

deployment to PaaS, 86–90, 103–109

development dependencies, 50–51

ActiveRecord model specs, 51–54

controller specs, 59–62

feature specs, 64–65

helper specs, 62–64

non-ActiveRecord model specs, 54–59

Devise, 29, 202

directories, deleting, 208

## E

engines

defined, 3

migrations, 28–34

models and, 142–146

routes

mounting, 24–25

testing, 60–62

ERD (entity-relationship diagram) in component extraction, 113–115

executing bundle, warnings during, 22

exposing requirable gem parts, 157

extracting components

bottom-up approach, 112–120

from existing applications, 196

Predictor component, 120–139

pulling up UI components, 150–167

pushing down model components, 168–183

top-down approach, 139–150

## F

files

deleting, 208

multiple updates, 204

renaming, 186–187

folders, renaming, 185–186

Foundation Responsive Library, 75

foundation-rails gem, 77–78

## G

games in Sportsball app

GamesAdmin

pulling up, 150–167  
UI dependencies, 142–150  
pushing down model components, 168–183

## gems

as common functionality components, 192–193  
creating, 120  
deleting files in, 121  
dependencies, 202–205  
exposing requirable parts, 157  
listing dependencies once, 98–99  
locking down versions, 38–40  
public gems, accidental usage, 135–136  
third-party service adapters, 191–192  
updating component versions, 93

## generating

AppComponent controller, 24  
components, 21, 151  
git submodules, 120–121  
global navigation in UI components, 165–167  
gradle, 238–241  
graphviz, 26

## H

healthy gemfiles, 94–95  
helper specs, 62–64  
Heroku, deployment to, 86–87, 103–109  
Hexagonal Architecture, 212–227

## I

initializers in persistence component, 207  
installing engine migrations with Rake, 28–30  
interaction (in DCI), 227

## J

Java, [13](#), [238–241](#)

Jenkins, [70–72](#)

## K

Kotlin, [238–241](#)

## L

loading engine migrations, [30–31](#)

locking down gem versions, [38–40](#)

long-running dependency updates, [97–98](#)

## M

maintenance, [8–9](#)

mapping dependencies in component refactoring, [140–141](#)

maven, [238](#)

Metcalfe’s law, [10](#), [11](#)

migrations

with Cloud Foundry, [89–90](#)

in component refactoring, [172–178](#), [201](#)

within components, [27–34](#)

missing templates, [36–37](#)

models

engines and, [142–146](#)

pushing down, [168–183](#)

uploaders, [201–202](#)

mounting routes, [24–25](#)

moving component files, [151–156](#)

## N

namespaced classes, testing, [63–64](#)

naming components, [20–21](#)

.NET, [241–243](#)

network complexities, 9–12  
non-ActiveRecord model specs, 54–59

## O

objects  
components versus, 1–2  
creating in tests, 57–58

## P

PaaS (Platforms-as-a-Service), deployment to, 86–90, 103–109

path blocks, dependencies with, 34–35

patterns, 13

persistence component, 200–208

predictions in Sportsball app, 43–48

PredictionController class

in DCI, 229–231  
in Hexagonal Architecture, 215–219  
testing, 59–62

PredictionHelper class, testing, 62–64

PredictionUI class

dependencies, 142–150  
pulling up, 150–167

Predictor class

component extraction, 120–139  
lack of dependencies, 142  
ratable objects, 231–232  
roles (in DCI), 233–234  
testing, 54–59

production-ready assets, 78–82

public gems, accidental usage, 135–136

public interface of classes, 51

Publify, 113–114

Publisher module, 216–217

pulling up UI components, 150–167  
pushing down model components, 168–183  
PWS (Pivotal Web Services), 86–90, 103–109

## R

Rails

architecture, 100  
Ruby versus, 13  
updating, 93–97

Rails ERD, 113

Rake, installing engine migrations, 28–30

ratable objects, 231–232

rating calculations, 40–43

Reed’s law, 10, 11

refactoring components

API components, 190–191  
common functionality components, 192–193  
from existing applications, 196–208  
extraction  
    bottom-up approach, 112–120  
    Predictor component, 120–139  
    pulling up UI components, 150–167  
    pushing down model components, 168–183  
    top-down approach, 139–150  
renaming, 183–188  
splitting, 190  
third-party service adapters, 191–192

RefineryCMS, 114–115

removing  
    dependencies in component refactoring, 128–131  
    Web framework, 101–102  
renaming components, 183–188

roles (in DCI), 227, 233–234

routes

engines

mounting, 24–25

testing, 60–62

UI components, testing, 159

RSpec, 50

Ruby, component-based, 13

runtime dependencies

locking down gem versions, 38–40

objects versus components, 1–2

with path blocks, 34–35

sample dependency graph, 5

slim-rails gem usage, 35–37

trueskill gem usage, 40–43

## S

scaffolds, advantages of, 28

scripting refactoring, 197–207

searcher methods in persistence component, 205–207

shoulda-matchers gem, 51–54

slim-rails gem, 35–37

source code

for model components, 182–183

obtaining, 18, 49–50

for UI components, 167

splitting components, 190

Sportsball app

AppComponent in, 23

component dependencies, 34–43

component extraction

bottom-up approach, 112–120

Predictor component, 120–139  
pulling up UI components, 150–167  
pushing down model components, 168–183  
top-down approach, 139–150  
DCI (data-context-integration) and, 229–234  
explained, 18  
Gemfile, 23–24  
Hexagonal Architecture and, 214–226  
migrations within components, 27–28  
predictions in, 43–48  
root folder, 99  
switching databases, 82–85  
testing components, 51–65  
sprockets 4, app folder in, 19  
structural software patterns, 13  
submodules, 120–121  
swappable data storage, 223–226  
switching databases, 82–85  
system setup, 17

## T

teams in Sportsball app  
data storage repository, 219–222  
pushing down model components, 168–183  
swappable data storage, 223–226  
TeamsAdmin  
pulling up, 150–167  
UI dependencies, 142–150  
templates, missing, 36–37  
testing  
components, 50–65, 170–183  
main application, 65–66

CI server setup, 70–74  
refactored components, 137–138  
shell scripts for, 67–70  
UI components, 160–165  
what to test, 66–67

namespaced classes, 63–64  
object creation in, 57–58  
refactored components, 122–138, 197–207  
UI components, 156–167

third-party service adapters, 191–192  
ticketee sample application, obtaining, 197  
top-down approach to component extraction, 139–150

## Travis CI

changing architecture, 102–103  
setting up, 72–74  
trueskill gem, 40–43

## U

UI components  
dependencies, 142–150  
pulling up, 150–167  
updating  
dependencies, 90–99  
files multiple times, 204  
uploaders for models, 201–202

## V

version control of refactoring script, 198–199

## W

Web framework, removing, 101–102



## Register Your Product at [informit.com/register](http://informit.com/register) Access additional benefits and **save 35%** on your next purchase

- Automatically receive a coupon for 35% off your next purchase, valid for 30 days. Look for your code in your InformIT cart or the Manage Codes section of your account page.
- Download available product updates.
- Access bonus material if available.\*
- Check the box to hear from us and receive exclusive offers on new editions and related products.

\*Registration benefits vary by product. Benefits will be listed on your account page under Registered Products.

---

### InformIT.com—The Trusted Technology Learning Source

InformIT is the online home of information technology brands at Pearson, the world's foremost education company. At InformIT.com, you can:

- Shop our books, eBooks, software, and video training
- Take advantage of our special offers and promotions ([informit.com/promotions](http://informit.com/promotions))
- Sign up for special offers and content newsletter ([informit.com/newsletters](http://informit.com/newsletters))
- Access thousands of free chapters and video lessons

Connect with InformIT—Visit [informit.com/community](http://informit.com/community)



Addison-Wesley • Adobe Press • Cisco Press • Microsoft Press • Pearson IT Certification • Prentice Hall • Que • Sams • Peachpit Press



# Code Snippets

---

```
$ rvm get stable
$ rvm install 2.4.2
$ gem install bundler -v '1.15.4'
$ gem install rails -v '5.1.4'
```

```
$ rails new sportsball
$ cd sportsball
```

```
$ rails plugin new components/app_component --full --mountable
```

```
$ bundle
The latest bundler is 1.16.0.pre.3, but you are currently running
1.15.4.
To update, run `gem install bundler --pre`
You have one or more invalid gemspecs that need to be fixed.
The gemspec at ./components/app_component/app_component.gemspec is
not valid.
Please fix this gemspec.
The validation error was '"FIXME" or "TODO" is not a description'
```

```
1 $:.push File.expand_path("../lib", __FILE__ )
2
3 # Maintain your gems version:
4 require "app_component/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name          = "app_component"
9   s.version       = AppComponent::VERSION
10  s.authors       = ["Stephan Hagemann"]
11  s.summary       = "Summary of AppComponent."
12  s.license        = "MIT"
13
14  s.files = Dir["{app,config,db,lib}/**/*",
15                "MIT-LICENSE", "Rakefile", "README.md"]
16
17  s.add_dependency "rails", "~> 5.1.0"
18
19 end
```

```
$ bundle
The latest bundler is 1.16.0.pre.3, but you are currently running
1.15.4.
To update, run `gem install bundler --pre`
Resolving dependencies...
Using rake 12.2.1

. . .

Using rails 5.1.4
Using sass-rails 5.0.6
Using app_component 0.1.0 from source at `components/app_component`
Bundle complete! 17 Gemfile dependencies, 73 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.
```

```
1 source 'https://rubygems.org'  
2  
3 . . .  
4  
5 gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]  
6 gem 'app_component', path: 'components/app_component'
```

```
$ cd components/app_component  
$ rails g controller welcome index
```

```
1 AppComponent::Engine.routes.draw do
2   root to: "welcome#index"
3 end
```

```
1 Rails.application.routes.draw do
2   mount AppComponent::Engine, at: "/"
3 end
```

```
$ rails s
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.10.0 (ruby 2.4.2-p198), codename: Russell's Teapot
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

```
$ brew install graphviz  
$ gem install cobradeps
```

```
1 gem 'app_component',
2     path: 'components/app_component',
3     group: [:default, :direct]
```

```
$ cobradeps -g component_diagram .
```

```
$ rails g scaffold team name:string  
  
$ rails g scaffold game date:datetime \  
    location:string \  
    first_team_id:integer \  
    second_team_id:integer \  
    winning_team:integer \  
    first_team_score:integer \  
    second_team_score:integer
```

```
$ rake db:migrate

== 20171029235211 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
 -> 0.0005s
== 20171029235211 CreateAppComponentTeams: migrated (0.0006s) =====

== 20171029235221 CreateAppComponentGames: migrating =====
-- create_table(:app_component_games)
 -> 0.0007s
== 20171029235221 CreateAppComponentGames: migrated (0.0007s) =====

$ cd ../..
$ rake db:migrate
Running via Spring preloader in process 58196
```

```
$ rake app_component:install:migrations
Running via Spring preloader in process 58464
Copied migration 20171030000159_create_app_component_teams.\
app_component.rb from app_component
Copied migration 20171030000160_create_app_component_games.\
app_component.rb from app_component
```

```
$ tree ./components/app_component/db/migrate
components/app_component/db/migrate
└── 20171029235211_create_app_component_teams.rb
    └── 20171029235221_create_app_component_games.rb
```

```
$ tree ./db/migrate
./db/migrate
└── 20170507205125_create_app_component_teams.app_component.rb
    └── 20170507205126_create_app_component_games.app_component.rb
```

```
1 module AppComponent
2   class Engine < ::Rails::Engine
3     isolate_namespace AppComponent
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11  end
```

```
1 Rake::Task["db:load_config"].enhance [:environment]
```

```
$ rake app_component:db:migrate
rake aborted!
ActiveRecord::DuplicateMigrationNameError:
Multiple migrations have the name CreateAppComponentTeams
```

```
$ rake db:migrate
Running via Spring preloader in process 58740
== 20171029235211 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
 -> 0.0010s
== 20171029235211 CreateAppComponentTeams: migrated (0.0010s) =====

== 20171029235221 CreateAppComponentGames: migrating =====
-- create_table(:app_component_games)
 -> 0.0007s
== 20171029235221 CreateAppComponentGames: migrated (0.0007s) =====

$ rails s
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.10.0 (ruby 2.4.2-p198), codename: Russell's Teapot
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

```
1 gem 'app_component', path: 'components/app_component'
```

```
1 path "components" do
2   gem "app_component"
3 end
```

```
1 path "components" do
2   gem "app_component"
3   gem "component_a"
4   gem "component_b"
5 end
```

```
1 s.add_dependency "slim-rails"
```

```
1 h1 Welcome to Sportsball!
2 p Predicting the outcome of matches since 2015.
3
4 = link_to "Manage Teams", teams_path
5 | &nbsp;|&nbsp;
6 = link_to "Manage Games", games_path
```

```
1 require "slim-rails"  
2  
3 module AppComponent  
4   require "app_component/engine"  
5 end
```

```
1 module AppComponent
2   class Engine < ::Rails::Engine
3     isolate_namespace AppComponent
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11
12    config.generators do |g|
13      g.orm           :active_record
14      g.template_engine :slim
15      g.test_framework :rspec
16    end
17  end
18 end
```

```
1 s.add_dependency "rails", "~> 5.1.4"
2 s.add_dependency "slim-rails"
```

```
1 s.add_dependency "rails", "5.1.4"
2 s.add_dependency "slim-rails", "3.1.3"
```

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 gem "trueskill",
6     git: "https://github.com/benjaminleesmith/trueskill",
7     ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

```
1 s.add_dependency "rails", "5.1.4"
2 s.add_dependency "slim-rails", "3.1.3"
3 s.add_dependency "trueskill"
```

```
$ bundle
The latest bundler is 1.16.0.pre.3, but you are currently running \
1.15.4.
To update, run `gem install bundler --pre`
Fetching https://github.com/benjaminleesmith/trueskill
Fetching gem metadata from https://rubygems.org/.....
Fetching version metadata from https://rubygems.org/..
Fetching dependency metadata from https://rubygems.org/..
Resolving dependencies...

...
Using trueskill 1.0.0 from https://github.com/benjaminleesmith/\
trueskill (at e404f45@e404f45)

...
Bundle complete! 3 Gemfile dependencies, 46 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.
```

```
1 gem "trueskill",
2   git: "https://github.com/benjaminleesmith/trueskill",
3   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

```
1 require "slim-rails"
2 require "saulabs/trueskill"
3
4 module AppComponent
5   require "app_component/engine"
6 end
```

```

1 module AppComponent
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7           rating: [Saulabs::TrueSkill::Rating.new(
8             1500.0, 1000.0, 1.0)]
9         }
10      memo
11    end
12  end
13
14  def learn(games)
15    games.each do |game|
16      first_team_rating =
17        @teams_lookup[game.first_team_id][:rating]
18      second_team_rating =
19        @teams_lookup[game.second_team_id][:rating]
20      game_result = game.winning_team == 1 ?
21        [first_team_rating, second_team_rating] :
22        [second_team_rating, first_team_rating]
23      Saulabs::TrueSkill::FactorGraph.new(
24        game_result, [1, 2]).update_skills
25    end
26  end
27
28  def predict(first_team, second_team)
29    team1 = @teams_lookup[first_team.id][:team]
30    team2 = @teams_lookup[second_team.id][:team]
31    winner = higher_mean_team(first_team, second_team) ?
32      team1 : team2
33    AppComponent::Prediction.new(team1, team2, winner)
34  end
35
36  def higher_mean_team(first_team, second_team)
37    @teams_lookup[first_team.id][:rating].first.mean >
38      @teams_lookup[second_team.id][:rating].first.mean
39  end
40 end
41 end

```

```
1 module AppComponent
2   class Prediction
3     attr_reader :first_team, :second_team, :winner
4
5     def initialize(first_team, second_team, winner)
6       @first_team = first_team
7       @second_team = second_team
8       @winner = winner
9     end
10    end
11  end
```

```
1 require_dependency "app_component/application_controller"
2 module AppComponent
3   class PredictionsController < ApplicationController
4     def new
5       @teams = AppComponent::Team.all
6     end
7
8     def create
9       predictor = Predictor.new(AppComponent::Team.all)
10      predictor.learn(AppComponent::Game.all)
11      @prediction = predictor.predict(
12        AppComponent::Team.find(params["first_team"]["id"]),
13        AppComponent::Team.find(params["second_team"]["id"]))
14     end
15   end
16 end
```

```
1 h1 Predictions
2
3 = form_tag prediction_path, method: "post" do |f|
4   .field
5     = label_tag :first_team_id
6     = collection_select(:first_team, :id, @teams, :id, :name)
7
8   .field
9     = label_tag :second_team_id
10    = collection_select(:second_team, :id, @teams, :id, :name)
11   .actions = submit_tag "What is it going to be?", class: "button"
```

```
1 h1 Prediction
2
3 =prediction_text @prediction.first_team, @prediction.second_team,
  @prediction.winner
4
5 .actions
6   = link_to "Try again!", new_prediction_path, class: "button"
```

```
1 module AppComponent
2   module PredictionsHelper
3     def prediction_text(team1, team2, winner)
4       "In the game between #{team1.name} and #{team2.name} " +
5       "the winner will be #{winner.name}"
6     end
7   end
8 end
```

```
1 h1 Welcome to Sportsball!
2 p Predicting the outcome of matches since 2015.
3
4 = link_to "Manage Teams", teams_path
5 | &nbsp;|&nbsp;
6 = link_to "Manage Games", games_path
7 | &nbsp;|&nbsp;
8 = link_to "Predict an outcome!", new_prediction_path
```

```
1 s.add_development_dependency "sqlite3"
2 s.add_development_dependency "rspec-rails"
```

```
1 RSpec.describe AppComponent::Game do
2   it { should validate_presence_of :date }
3   it { should validate_presence_of :location }
4   it { should validate_presence_of :first_team }
5   it { should validate_presence_of :second_team }
6   it { should validate_presence_of :winning_team }
7   it { should validate_presence_of :first_team_score }
8   it { should validate_presence_of :second_team_score }
9
10  it { should belong_to :first_team}
11  it { should belong_to :second_team}
12 end
```

```
1 module AppComponent
2   class Game < ApplicationRecord
3     validates :date, :location, :first_team, :second_team,
4               :winning_team, :first_team_score, :second_team_score,
5               presence: true
6     belongs_to :first_team, class_name: "Team"
7     belongs_to :second_team, class_name: "AppComponent::Team"
8   end
9 end
```

```
$ rspec spec
./components/app_component/spec/controllers/app_component/\
games_controller_spec.rb:1:in `<top (required)>':
  uninitialized constant AppComponent::GamesController (NameError)
```

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10 require "ostruct"
11
12 require "rails-controller-testing"
13 Rails::Controller::Testing.install
14
15 Dir[AppComponent::Engine.root.join("spec/support/**/*.rb")].
16   each { |f| require f }
17
18 RSpec.configure do |config|
19   config.expect_with :rspec do |expectations|
20     expectations.
21       include_chain_clauses_in_custom_matcher_descriptions = true
22   end
23   config.mock_with :rspec do |mocks|
24     mocks.verify_partial_doubles = true
25   end
26
27 config.infer_spec_type_from_file_location!
28 config.disable_monkey_patching!
29 config.warnings = false
30 config.profile_examples = nil
31 config.order = :random
32 Kernel.srand config.seed
33
34 config.before(:suite) do
35   DatabaseCleaner.strategy = :transaction
36   DatabaseCleaner.clean_with(:truncation)
37 end
38
39 config.around(:each) do |example|
40   DatabaseCleaner.cleaning do
41     example.run
42   end
43 end
44
45 [:controller, :view, :request].each do |type|
```

```
46     config.include ::Rails::Controller::Testing::
47         TestProcess, :type => type
48     config.include ::Rails::Controller::Testing::
49         TemplateAssertions, :type => type
50     config.include ::Rails::Controller::Testing::
51         Integration, :type => type
52 end
53
54 config.include ObjectCreationMethods
55 end
56
57 Shoulda::Matchers.configure do |config|
58   config.integrate do |with|
59     with.test_framework :rspec
60     with.library :rails
61   end
62 end
```

```
1 RSpec.describe AppComponent::Predictor do
2   before do
3     @team1 = create_team name: "A"
4     @team2 = create_team name: "B"
5
6     @predictor = AppComponent::Predictor.new([\@team1, \@team2])
7   end
8
9   it "predicts teams that have won in the past to win in the
10    future" do
11     game = create_game first_team: @team1,
12                         second_team: @team2, winning_team: 1
13     @predictor.learn([game])
14
15     prediction = @predictor.predict(@team2, @team1)
16     expect(prediction.winner).to eq @team1
17
18     prediction = @predictor.predict(@team1, @team2)
19     expect(prediction.winner).to eq @team1
20   end
21
22   it "changes predictions based on games learned" do
23     game1 = create_game first_team: @team1,
24                           second_team: @team2, winning_team: 1
25     game2 = create_game first_team: @team1,
26                           second_team: @team2, winning_team: 2
27     game3 = create_game first_team: @team1,
28                           second_team: @team2, winning_team: 2
29     @predictor.learn([game1, game2, game3])
30
31     prediction = @predictor.predict(@team1, @team2)
32     expect(prediction.winner).to eq @team2
33   end
34
35   it "behaves funny when teams are equally strong" do
```

```
36 prediction = @predictor.predict(@team1, @team2)
37 expect(prediction).to be_an AppComponent::Prediction
38 expect(prediction.first_team).to eq @team1
39 expect(prediction.second_team).to eq @team2
40 expect(prediction.winner).to eq @team2
41
42 prediction = @predictor.predict(@team2, @team1)
43 expect(prediction).to be_an AppComponent::Prediction
44 expect(prediction.first_team).to eq @team2
45 expect(prediction.second_team).to eq @team1
46 expect(prediction.winner).to eq @team1
47 end
43 end
```

```

1 module AppComponent
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7           rating: [
8             Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
9           ]
10        }
11        memo
12      end
13    end
14
15    def learn(games)
16      games.each do |game|
17        first_team_rating = @teams_lookup[game.first_team_id] [
18          :rating]
19        second_team_rating = @teams_lookup[game.
20          second_team_id] [:rating]
21        game_result = game.winning_team == 1 ?
22          [first_team_rating, second_team_rating] :
23          [second_team_rating, first_team_rating]
24        Saulabs::TrueSkill::FactorGraph.new(game_result, [1, 2])..
25          update_skills
26      end
27    end
28
29    def predict(first_team, second_team)
30      team1 = @teams_lookup[first_team.id] [:team]
31      team2 = @teams_lookup[second_team.id] [:team]
32      winner = higher_mean_team(first_team, second_team) ?
33        team1 : team2
34      AppComponent::Prediction.new(team1, team2, winner)
35    end
36
37    def higher_mean_team(first_team, second_team)
38      @teams_lookup[first_team.id] [:rating].first.mean >
39        @teams_lookup[second_team.id] [:rating].first.mean
40    end
41  end
42 end

```

```
1 module ObjectCreationMethods
2   def new_team(overrides = {})
3     defaults = {
4       name: "Some name #{counter}"
5     }
6     AppComponent::Team.new {
7       |team| apply(team, defaults, overrides)
8     }
9   end
10
11  def create_team(overrides = {})
12    new_team(overrides).tap(&:save!)
13  end
14
15  def new_game(overrides = {})
16    defaults = {
17      first_team: -> { new_team },
18      second_team: -> { new_team },
19      winning_team: 2,
20      first_team_score: 2,
21      second_team_score: 3,
22      location: "Somewhere",
23      date: Date.today
24    }
25
26  AppComponent::Game.new {
27    |game| apply(game, defaults, overrides)
28  }
```

```
29   end
30
31   def create_game(overrides = {})
32     new_game(overrides).tap(&:save!)
33   end
34
35   private
36
37   def counter
38     @counter ||= 0
39     @counter += 1
40   end
41
42   def apply(object, defaults, overrides)
43     options = defaults.merge(overrides)
44     options.each do |method, value_or_proc|
45       object. __send__ (
46         "#{method}=",
47         value_or_proc.is_a?(Proc) ?
48           value_or_proc.call : value_or_proc)
49     end
50   end
51 end
```

```
1 RSpec.describe AppComponent::PredictionsController,
2   :type => :controller do
3   routes { AppComponent::Engine.routes }
4
5   before do
6     @team1 = create_team
7     @team2 = create_team
8   end
9
10  describe "GET new" do
11    it "assigns all teams as @teams" do
12      get :new, params: {}, session: {}
13      expect(assigns(:teams)).to eq [@team1, @team2]
14    end
15  end
16
17  describe "POST create" do
18    it "assigns a prediction as @prediction" do
19      post :create,
20        params: {
21          first_team: {id: @team1.id},
22          second_team: {id: @team2.id}
23        },
24        session: {}
25
26      prediction = assigns(:prediction)
27      expect(prediction).to be_a AppComponent::Prediction
28      expect(prediction.first_team).to eq @team1
29      expect(prediction.second_team).to eq @team2
30
31    end
32  end
33 end
```

```
1 require_dependency "app_component/application_controller"
2 module AppComponent
3   class PredictionsController < ApplicationController
4     def new
5       @teams = AppComponent::Team.all
6     end
7
8     def create
9       predictor = Predictor.new(AppComponent::Team.all)
10      predictor.learn(AppComponent::Game.all)
11      @prediction = predictor.predict(
12        AppComponent::Team.find(params["first_team"]["id"]),
13        AppComponent::Team.find(params["second_team"]["id"]))
14    end
15  end
16 end
```

```
$ rspec spec
```

```
...
```

```
Failure/Error: get :new, {}, {}
ActionController::UrlGenerationError:
  No route matches {:action=>"new",
    :controller=>"app_component/predictions"}
```

```
1 module EngineControllerTestMonkeyPatch
2   def get(action, parameters = nil, session = nil, flash = nil)
3     process_action(action, parameters, session, flash, "GET")
4   end
5
6   def post(action, parameters = nil, session = nil, flash = nil)
7     process_action(action, parameters, session, flash, "POST")
8   end
9
10  def put(action, parameters = nil, session = nil, flash = nil)
11    process_action(action, parameters, session, flash, "PUT")
12  end
13
14  def delete(action, parameters = nil, session = nil, flash = nil)
15    process_action(action, parameters, session, flash, "DELETE")
16  end
17
18  private
19
20  def process_action(action, parameters = nil, session = nil,
21                     flash = nil, method = "GET")
22    parameters ||= {}
23    process(action,
24            method,
25            parameters.merge!(:use_route => :app_component),
26            # :app_component == engine name
27            session,
28            flash)
29  end
30 end
31
32 AppComponent::Engine.routes.
33   default_url_options = {host: "test.host"}
34 EngineRoutes = AppComponent::Engine.routes.url_helpers
35
36 RSpec.configure do |config|
37   config.include EngineControllerTestMonkeyPatch,
38     :type => :controller
39 end
```

```
1 require "spec_helper"
2
3 RSpec.describe AppComponent::PredictionsHelper, :type => :helper do
4   it "returns a nice prediction text" do
5     Named = Struct.new(:name)
6     text = prediction_text(
7       Named.new("A"), Named.new("B"), Named.new("C"))
8     expect(text).to eq
9       "In the game between A and B the winner will be C"
10   end
11 end
```

```
1 module AppComponent
2   module PredictionsHelper
3     def prediction_text(team1, team2, winner)
4       "In the game between #{team1.name} and #{team2.name} " +
5       "the winner will be #{winner.name}"
6     end
7   end
8 end
```

```
1 module AppComponent
2   RSpec.describe PredictionsHelper, :type => :helper do
3     it "returns a nice prediction text" do
4       Named = Struct.new(:name)
5       text = prediction_text(
6         Named.new("A"), Named.new("B"), Named.new("C"))
7       expect(text).to eq
8         "In the game between A and B the winner will be C"
9     end
10   end
11 end
```

```
1 module PredictionsHelper
2   def prediction_text(team1, team2, winner)
3     "In the game between #{team1.name} and #{team2.name} " +
4     "the winner will be #{winner.name}"
5   end
6 end
```

```
1 require "spec_helper"
2
3 RSpec.describe "the prediction process", :type => :feature do
4   before :each do
5     team1 = create_team name: "UofL"
6     team2 = create_team name: "UK"
7
8     create_game first_team: team1,
9                   second_team: team2, winning_team: 1
10    create_game first_team: team2,
11                  second_team: team1, winning_team: 2
12    create_game first_team: team2,
13                  second_team: team1, winning_team: 2
14  end
15
16  it "get a new prediction" do
17    visit "/app_component/"
18
19    click_link "Predictions"
20
21    select "UofL", from: "First team"
22    select "UK", from: "Second team"
23    click_button "What is it going to be"
24
25    expect(page).to have_content "the winner will be UofL"
26  end
27 end
```

```
1 require "spec_helper"
2
3 RSpec.describe "the app", :type => :feature do
4   it "hooks up to /" do
5     visit "/"
6     within "main h1" do
7       expect(page).to have_content "Sportsball"
8     end
9   end
10
11  it "has teams" do
12    visit "/"
13    click_link "Teams"
14    within "main h1" do
15      expect(page).to have_content "Teams"
16    end
17  end
18
19  it "has games" do
20    visit "/"
21    click_link "Games"
22    within "main h1" do
23      expect(page).to have_content "Games"
24    end
25  end
26
27  it "can predict" do
28    AppComponent::Team.create! name: "UofL"
29    AppComponent::Team.create! name: "UK"
30
31    visit "/"
32    click_link "Predictions"
33    click_button "What is it going to be"
34  end
35 end
```

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "**** Running app component engine specs"
6 bundle
7 bundle exec rake db:create db:migrate
8 RAILS_ENV=test bundle exec rake db:create db:migrate
9 bundle exec rspec spec
10 exit_code+=$?
11
12 exit $exit_code
```

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running container app specs"
6 bundle
7 bundle exec rake db:drop
8 bundle exec rake environment db:create db:migrate
9 RAILS_ENV=test bundle exec rake environment db:create db:migrate
10 bundle exec rspec spec
11 exit_code+=$?
12
13 exit $exit_code
```

```
1 #!/bin/bash
2
3 result=0
4
5 cd "$( dirname "${BASH_SOURCE[0]}" )"
6
7 for test_script in $(find . -name test.sh); do
8     pushd `dirname $test_script` > /dev/null
9     ./test.sh
10    ((result+= $?))
11    popd > /dev/null
12 done
13
14 if [ $result -eq 0 ]; then
15     echo "SUCCESS"
16 else
17     echo "FAILURE"
18 fi
19
20 exit $result
```

```
$ ./build.sh
*** Running app component engine specs
.....
```

```
Finished in 0.91921 seconds (files took 1.65 seconds to load)
70 examples, 0 failures
```

```
Randomized with seed 6563
```

```
*** Running container app specs
```

```
db/test.sqlite3 already exists
```

```
.
```

```
Finished in 0.07369 seconds (files took 1.56 seconds to load)
1 example, 0 failures
```

```
Randomized with seed 29486
```

```
SUCCESS
```

```
1 #!/bin/bash --login
2 export PATH="$PATH:$HOME/.rvm/bin" # Add RVM to PATH for scripting
3 rvm use "2.4.2@sportsball"
4
5 gem install bundler
6 ./build.sh
```

```
1 rvm:  
2   - 2.4.2  
3 script: ./build.sh
```

```
*** Running app component engine specs
...
/gems/activesupport-4.1.9/lib/active_support/dependencies.rb:247:in \
`require': cannot load such file -- shoulda/matchers (LoadError)
```

```
1 gemfile:
2   - Gemfile
3   - components/app_component/Gemfile
4 script:
5   - travis_retry ./test.sh
6 before_install:
7   - cd $(dirname $BUNDLE_GEMFILE)
8 rvm:
9   - 2.4.2
```

```
./components/app_component/vendor
└ assets
    └─ javascripts
        └─ app_component
            ├ fastclick.js
            ├ foundation.min.js
            ├ jquery.cookie.js
            ├ modernizr.js
            └ placeholder.js
    └ stylesheets
        └─ app_component
            ├ foundation.min.css
            └ normalize.css
```

```
1 //!= require app_component/modernizr.js
2 //!= require jquery
3 //!= require app_component/jquery.cookie.js
4 //!= require app_component/placeholder.js
5 //!= require app_component/fastclick.js
6 //!= require app_component/foundation.min.js
7 //!= require jquery_ujs
8
9 $(document).ready(function() {
10     $(document).foundation();
11});
```

```
1 /*  
2  *= require app_component/normalize  
3  *= require app_component/foundation.min  
4  *= require_self  
5 */
```

```
1 | <!DOCTYPE html>
2 html
3   head
4     meta charset="utf-8"
5     meta name="viewport" content="width=device-width,
6           initial-scale=1.0"
7   title Sportsball App
8
9   = javascript_include_tag "app_component/application"
10  = stylesheet_link_tag    "app_component/application",
11    media: "all"
12
13  = csrf_meta_tags
14
15 header
16   .contain-to-grid.sticky
17     nav.top-bar data-topbar="" role="navigation"
18       ul.title-area
19         li.name
20           h1
21             = link_to root_path do
22               = image_tag "app_component/logo.png", width: "25px"
23               | Predictor
24
25         li.toggle-topbar.menu-icon
26           a href="#"
27             span Menu
28
29         section.top-bar-section
30           ul.left
31             li =link_to "Teams", teams_path
32             li =link_to "Games", games_path
33             li =link_to "Predictions", new_prediction_path
34
35 main
36   .row
37     .small-12.columns
38       = yield
```

```
export SECRET_KEY_BASE=something_to_test
```

```
RAILS_ENV=production rake db:create  
RAILS_ENV=production rake db:migrate
```

```
$ RAILS_ENV=production rake assets:precompile
I, [2015-02-22T20:45:47.062189 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/\
logo-14e0571820ed8569b86519648f035ddd.png
I, [2015-02-22T20:45:49.695917 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/\
application-6e039fbe90e5f75b1b3a88842d1face8.js
I, [2015-02-22T20:45:50.829624 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/\
application-097b9ea4ccfcfd13ae1d38051c047dc0d.css
```

```
$ RAILS_ENV=production rake assets:precompile
I, [2015-02-22T20:45:47.062189 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/
logo-14e0571820ed8569b86519648f035ddd.png
```

```
1 Rails.application.config.assets.precompile += [  
2     'app_component/app_component_engine.js',  
3     'app_component/app_component_engine.css'  
4 ]
```

```
$ RAILS_ENV=production rake assets:precompile
I, [2015-02-22T20:45:47.062189 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/\
logo-14e0571820ed8569b86519648f035ddd.png
I, [2015-02-22T20:45:49.695917 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/\
app_engine-6e039fbe90e5f75b1b3a88842d1face8.js
I, [2015-02-22T20:45:50.829624 #55331] INFO -- : Writing
./sportsball/public/assets/app_component/\
app_engine-097b9ea4ccfcfd13ae1d38051c047dc0d.css
```

```
public/assets/
└── app_component
    ├── app_component_engine-097b9ea4ccfcfd13ae1d38051c047dc0d.css
    ├── app_component_engine-097b9ea4ccfcfd13ae1d38051c047dc0d.css.gz
    ├── app_component_engine-6e039fbe90e5f75b1b3a88842d1face8.js
    ├── app_component_engine-6e039fbe90e5f75b1b3a88842d1face8.js.gz
    └── logo-14e0571820ed8569b86519648f035ddd.png
└── manifest-7a9f0159d72e2415368891385ba14b40.json
```

```
1 source "https://rubygems.org"
2
3 git_source(:github) do |repo_name|
4   repo_name =
5     "#{repo_name}/#{repo_name}" unless repo_name.include?("/")
6   "https://github.com/#{repo_name}.git"
7 end
8
9 path "components" do
10   gem "app_component"
11 end
12
13 gem "trueskill",
14   git: "https://github.com/benjaminleesmith/trueskill",
15   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
16
17 gem "rails", "~> 5.1.4"
18 gem "pg", "0.21.0"
19 gem "puma", "~> 3.7"
20 gem "sass-rails", "~> 5.0"
21 gem "uglifier", ">= 1.3.0"
22
23 gem "coffee-rails", "~> 4.2"
24 gem "turbolinks", "~> 5"
25 gem "jbuilder", "~> 2.5"
26
27 group :development, :test do
28   gem "rspec-rails"
29   gem "byebug", platforms: [:mri, :mingw, :x64_mingw]
30   gem "capybara", "~> 2.13"
31   gem "selenium-webdriver"
32 end
33
34 group :development do
35   gem "web-console", ">= 3.3.0"
36   gem "listen", ">= 3.0.5", "< 3.2"
37   gem "spring"
38   gem "spring-watcher-listen", "~> 2.0.0"
39 end
40
41 gem "tzinfo-data", platforms: [:mingw, :mswin, :x64_mingw, :jruby]
```

```
1 development: &DEVELOPMENT
2   adapter: postgresql
3   database: sportsball_development
4   host: localhost
5   pool: 5
6   timeout: 5000
7
8 test: &TEST
9   <<: *DEVELOPMENT
10  database: sportsball_test
11  min_messages: warning
12
13 production:
14   adapter: postgresql
15   database: sportsball_production
```

```
1 $:.push File.expand_path("../lib", __FILE__ )
2
3 # Maintain your gems version:
4 require "app_component/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name          = "app_component"
9   s.version       = AppComponent::VERSION
10  s.authors       = ["Stephan Hagemann"]
11  s.email         = ["stephan.hagemann@gmail.com"]
12  s.homepage      = ""
13  s.summary        = "Summary of AppComponent."
14  s.description    = "Description of AppComponent."
15  s.license        = "MIT"
16
17  s.files = Dir["{app,config,db,lib}/**/*",
18    "MIT-LICENSE", "Rakefile", "README.md"]
19
20  s.add_dependency "rails", "5.1.4"
21  s.add_dependency "slim-rails", "3.1.3"
22  s.add_dependency "jquery-rails", "4.3.1"
23  s.add_dependency "trueskill"
24
25  s.add_development_dependency "pg"
26  s.add_development_dependency "rspec-rails"
27  s.add_development_dependency "shoulda-matchers"
28  s.add_development_dependency "database_cleaner"
29  s.add_development_dependency "capybara"
30  s.add_development_dependency "rails-controller-testing"
31 end
```

```
1 development: &DEVELOPMENT
2   adapter: postgresql
3   database: sportsball_app_component_development
4   host: localhost
5   pool: 5
6   timeout: 5000
7
8 test: &TEST
9   <<: *DEVELOPMENT
10  database: sportsball_app_component_test
11  min_messages: warning
```

```
$ git push heroku master  
$ heroku run rake db:migrate
```

```
$ cf login -a https://api.run.pivotal.io  
      #follow the instructions on screen  
$ cf create-service elephantsql turtle sportsball_db  
$ cf push sportsball --no-start  
$ cf bind-service sportsball sportsball_db  
$ cf start sportsball
```

```
$ cf env sportsball
Getting env variables for app sportsball in
    org sportsball / space development as shagemann@pivotal.io...
OK
```

System-Provided:

```
{
  "VCAP_SERVICES": {
    "elephantsql": [
      {
        "credentials": {
          "max_conns": "5",
          "uri": "postgres://rfghjuyt:qwertyuioplkjhgfdsa@babar
                  .elephantsql.com:5432/rfghjuyt"
        },
        "label": "elephantsql",
        "name": "sportsball",
        "plan": "turbo",
        "tags": [
          "Data Stores",
          "Cloud Databases",
          "Developer Tools",
          "Data Store",
          "postgresql",
          "relational",
          "New Product"
        ]
      }
    ]
  }
}
```

No user-defined env variables have been **set**

```
1 production:
2   adapter: postgresql
3   database: rfghjuyt
4   username: rfghjuyt
5   password: qwertyuioplkjhgfdsa
6   host: babar.elephantsql.com
```

```
$ RAILS_ENV=production rake db:migrate
Migrating to CreateAppComponentTeams (20150110181102)
== 20150110181102 CreateAppComponentTeams: migrating =====
-- create_table(:app_teams)
 -> 0.0945s
== 20150110181102 CreateAppComponentTeams: migrated (0.0947s) =====

Migrating to CreateAppComponentGames (20150110184219)
== 20150110184219 CreateAppComponentGames: migrating =====
-- create_table(:app_games)
 -> 0.0722s
== 20150110184219 CreateAppComponentGames: migrated (0.0724s) =====
```

```
1 #!/bin/bash
2
3 # BUNDLE UPDATE ENHANCER
4 # Preconditions:
5 # all components must have a Gemfile
6 # all components must use RVM and
7 #have a .ruby-version and .ruby-gemset
8
9 if [ $# -eq 0 ]; then
10    echo "No arguments supplied!"
11    echo "You must provide the name of the gem you want to update" +
12        "in your main app and all your components."
13    exit
14 fi
15
16 # ensure we bundle first
17 bundle
18
19 bundle show $1 > /dev/null
20 if [ $? -eq 0 ]; then
21
22    echo ">> BUE found $1 inside your main application. " +
23        "Updating it and its components now."
24    bundle
25    bundle update $1
26
27    echo ">> searching $1 inside the components"
28
29 # Iterate over the existing components
30 for component in components/* ; do
31
32    # Change directory to examined component
33    pushd $component > /dev/null
34
35    # Enable rvm
```

```
36 source "$HOME/.rvm/scripts/rvm"
37 rvm use $(cat .ruby-version)@$$(cat .ruby-gemset) --create \
38 > /dev/null
39
40 # ensure we bundle first
41 bundle
42 # bundle show is a good citizen and
43 # returns unix code 0 when successful.
44 bundle show $1 > /dev/null
45 if [ $? -eq 0 ]; then
46     echo ">> BUE found $1 inside $component. Updating it now."
47     bundle update $1
48 fi
49
50 # Move back directory.
51 popd > /dev/null
52 done
53
54 echo ">> All done"
55 else
56     echo "?? Nothing done, could not find $1"
57 fi
```

```
1 for component_file in $(find . -name *.gemspec); do  
2     component=`dirname $component_file`
```

```
$ bundle update rails
Fetching gem metadata from https://rubygems.org/.....
Fetching version metadata from https://rubygems.org/...
Fetching dependency metadata from https://rubygems.org/..
Resolving dependencies.....
Bundler could not find compatible versions for gem "actionmailer":
  In Gemfile:
    rails (= 5.0.0.1) was resolved to 5.0.0.1, which depends on
      actionmailer (= 5.0.0.1)

    app_component was resolved to 0.0.1, which depends on
      slim-rails (= 3.0.1) was resolved to 3.0.1, which depends on
        actionmailer (< 5.0, >= 3.1)
```

```
1 RSpec.configure do |config|
2
3   ...
4
5   [:controller, :view, :request].each do |type|
6     config.include ::Rails::Controller::Testing::TestProcess,
7       :type => type
8     config.include ::Rails::Controller::Testing::TemplateAssertions,
9       :type => type
10    config.include ::Rails::Controller::Testing::Integration,
11      :type => type
12  end
13 end
```

1) the prediction process get a new prediction  
Failure/Error: visit '/app\_component/'  
ActionView::Template::Error:  
Asset was not declared to be precompiled in production.  
Add `Rails.application.config.assets.precompile +=  
%w( app\_component/logo.png )` to  
`config/initializers/assets.rb` and restart your server

```
1 module AppComponent
2   class Engine < ::Rails::Engine
3     isolate_namespace AppComponent
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11
12    initializer :assets do |app|
13      app.config.assets.precompile += %w( app_component/logo.png )
14    end
15
16    config.generators do |g|
17      g.orm             :active_record
18      g.template_engine :slim
19      g.test_framework  :rspec
20    end
21  end
22 end
```

```
1 if ENV['RAILS_5_UPDATE']
2   s.add_dependency "rails", "5.0.0.1"
3 else
4   s.add_dependency "rails", "4.1.9"
5 end
```

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running app component engine specs FOR RAILS 5"
6 RAILS_5_UPDATE=true bundle exec rake db:create db:migrate
7 RAILS_5_UPDATE=true RAILS_ENV=test bundle exec \
8   rake db:create db:migrate
9 RAILS_5_UPDATE=true bundle exec rspec spec
10 exit_code+=$?
11
12 exit $exit_code
```

```
./sportsball
└ components
  └ app_component
    └ ...
  └ rails_version
    └ rails_version.gemspec
```

```
1 $:.push File.expand_path("../lib", __FILE__ )
2
3 # Describe your gem and declare its dependencies:
4 Gem::Specification.new do |s|
5   s.name          = "rails_version"
6   s.version       = "0.0.1"
7   s.authors       = ["The CBRA Book"]
8   s.summary        = "CBRA component"
9
10  s.files = []
11  s.test_files = []
12
13  s.add_dependency "rails", "5.1.4"
14 end
```



```
sportsball
├── bin
├── components
├── config
├── db
├── lib
├── log
├── public
├── spec
├── tmp
└── vendor
    ├── Gemfile
    ├── Gemfile.lock
    ├── README.md
    ├── Rakefile
    ├── build.sh
    ├── config.ru
    ├── package.json
    ├── test.sh
    └── vendor.tree
```

```
$ mkdir web_container  
$ mv * web_container  
$ mv web_container/components .
```



sportsball

```
└── components
    └── app_component
        └── web_container
            ├── bin
            ├── config
            ├── db
            ├── lib
            ├── log
            ├── public
            ├── spec
            ├── tmp
            ├── vendor
            ├── Gemfile
            ├── Gemfile.lock
            ├── README.md
            ├── Rakefile
            ├── build.sh
            ├── config.ru
            ├── package.json
            ├── test.sh
            └── vendor.tree
```

```
$ rails s  
The path `./web_container/components/app_component` does not exist.
```

```
1 gemfile:
2   - web_container/Gemfile
3   - components/app_component/Gemfile
4 script:
5   - ./test.sh
6 before_install:
7   - cd $(dirname $BUNDLE_GEMFILE)
8 rvm:
9   - 2.4.2
```

Failures:

1) Engine 'AppComponent' hooks up to /  
Failure/Error: visit "/"

NoMethodError:

```
undefined method `visit' for  
#<RSpec::ExampleGroups::EngineApp:0x00000002524390>  
# ./spec/features/app_spec.rb:3:in  
# `block (2 levels) in <top (required)>'
```

Finished in 0.00062 seconds (files took 0.21662 seconds to load)

1 example, 1 failure

Failed examples:

```
rspec ./spec/features/app_spec.rb:2
```

```
# Engine 'AppComponent' hooks up to /
```

The command "./test.sh" exited with 1.

Done. Your build exited with 1.

```
$ cf push sportsball
Updating app sportsball in org sportsball / space development
    as stephan...
OK
```

```
Uploading sportsball...
Uploading app files from: ./sportsball
Uploading 4.6M, 922 files
OK
```

```
Stopping app sportsball in org sportsball / space development
    as stephan...
OK
```

```
Starting app sportsball in org sportsball / space development
    as stephan...
```

```
OK
-----> Downloaded app package (4.9M)
```

```
FAILED
Server error, status code: 400, error code: 170003, message:
An app was not successfully detected by any available buildpack
```

```
1 #!/bin/bash
2 trap "exit" ERR
3
4 echo "      Copy deploy files into place"
5 rm -rf deploy
6 mkdir deploy
7 cp -R web_container/ deploy
8 cp -R components deploy
9 rm -rf deploy/tmp/*
10
11 echo "      Fix components directory reference"
12 sed -i -- s|\"..../components|"components|g deploy/Gemfile
13 sed -i -- s|remote: ..../components|remote: ..../components|g \
14     deploy/Gemfile.lock
15
16 echo "      Uploading application...."
```

```
1 #!/bin/bash
2 trap "exit" ERR
3
4 APP_NAME="sportsball"
5
6 echo "-----> Deploying to Cloud Foundry"
7 prepare_deploy_directory.sh
8
9 cd deploy
10 cf push $APP_NAME
11 cd ..
```

```
Running: bundle install --without development:test  
--path vendor/bundle --binstubs vendor/bundle/bin -j4  
--deployment  
You are trying to install in deployment mode after changing  
your Gemfile. Run `bundle install` elsewhere and add the  
updated Gemfile.lock to version control.  
You have added to the Gemfile:  
* source: source at ../components/app_component  
You have deleted from the Gemfile:  
* source: source at vendor/cache/app_component
```

```
1 PATH
2   remote: vendor/cache/app_component
3   specs:
4     app_component (0.1.0)
```

```
1 gem 'app_component', path: '../components/app_component'
```

```
1 gem 'app_component',
2     path: (Pathname.new('..../components').exist? ?
3             '..../component\
3 s/app_component' :
4     'vendor/cache/app_component')
```

```
1 #!/bin/bash
2 trap "exit" ERR
3
4 APP_NAME="stormy-hollows-9630"
5
6 echo "-----> Deploying to Heroku"
7 prepare_deploy_directory.sh
8
9 VERSION=`git rev-parse HEAD | perl -pe "chomp"`
10 echo "-----> Deploying application version $VERSION"
11
12 echo "      Creating build tarball...."
13 DEPLOY_FILENAME="deploy-$VERSION.tgz"
14 pushd deploy
15 tar -czf ../$DEPLOY_FILENAME .
16 popd
17
18 echo "      Requesting application specific source endpoint..."
19 acceptheader="Accept: application/vnd.heroku+json; "
20 acceptheader+="version=3.streaming-build-output)"
21 SOURCE_ENDPOINT=$(curl -s -n \
22     -X POST "https://api.heroku.com/apps/$APP_NAME/sources" \
23     -H $acceptheader)"
24
25 PUT_URL=`echo $SOURCE_ENDPOINT | jsawk \
26     "return this.source_blob.put_url"`
27 echo "      Received blob endpoint: $PUT_URL"
28 GET_URL=`echo $SOURCE_ENDPOINT | jsawk \
29     "return this.source_blob.get_url"`
30 echo "      Received deploy endpoint: $GET_URL"
31
32 echo "      Upload app blob"
33 curl -s "$PUT_URL" -X PUT -H "Content-Type:" \
34     --data-binary @$DEPLOY_FILENAME
35
36 echo "      Deploy application"
37
38 data='{"source_blob": {"url": "$GET_URL", " "
39 data+=" "version": "$VERSION"} }'
40 DEPLOY_RESULT=$(curl -n
41     -X POST "https://api.heroku.com/apps/$APP_NAME/builds" \
42     -d $data
43     -H $acceptheader -H "Content-Type: application/json")"
```

```
44
45 log_url=`echo "$DEPLOY_RESULT" | \
46     jsawk "return this.output_stream_url"`
47 echo "        Received log endpoint: $log_url"
48
49 curl "$log_url"
```

```
./sportsball/components/app_component/app
├── assets
│   ├── config
│   │   └── app_component_manifest.js
│   ├── images
│   │   └── app_component
│   │       └── logo.png
│   ├── javascripts
│   │   └── app_component
│   │       └── application.js
│   └── stylesheets
│       └── app_component
│           └── application.css
└── controllers
    └── app_component
        └── application_controller.rb
            ├── games_controller.rb
            ├── predictions_controller.rb
            ├── teams_controller.rb
            └── welcome_controller.rb
├── helpers
│   └── app_component
│       └── predictions_helper.rb
└── models
    └── app_component
        ├── application_record.rb
        ├── game.rb
        ├── prediction.rb
        ├── predictor.rb
        └── team.rb
└── views
```

```
└── app_component
    ├── games
    │   ├── _form.html.slim
    │   ├── edit.html.slim
    │   ├── index.html.slim
    │   ├── new.html.slim
    │   └── show.html.slim
    ├── predictions
    │   ├── create.html.slim
    │   └── new.html.slim
    ├── teams
    │   ├── _form.html.slim
    │   ├── edit.html.slim
    │   ├── index.html.slim
    │   └── new.html.slim
    └── welcome
        └── show.html.slim
layouts
└── app_component
    └── application.html.slim
```

```
1 module AppComponent
2   module PredictionsHelper
3     def prediction_text(team1, team2, winner)
4       "In the game between #{team1.name} and #{team2.name} " +
5       "the winner will be #{winner.name}"
6     end
7   end
8 end
```

```
1 module AppComponent
2   class Prediction
3     attr_reader :first_team, :second_team, :winner
4
5     def initialize(first_team, second_team, winner)
6       @first_team = first_team
7       @second_team = second_team
8       @winner = winner
9     end
10    end
11  end
```

```

1 module AppComponent
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7           rating: [
8             Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
9           ]
10        }
11      memo
12    end
13  end
14
15  def learn(games)
16    games.each do |game|
17      first_team_rating =
18        @teams_lookup[game.first_team_id][:rating]
19      second_team_rating =
20        @teams_lookup[game.second_team_id][:rating]
21      game_result = game.winning_team == 1 ?
22        [first_team_rating, second_team_rating] :
23        [second_team_rating, first_team_rating]
24      Saulabs::TrueSkill::FactorGraph.
25        new(game_result, [1, 2]).update_skills
26    end
27  end
28
29  def predict(first_team, second_team)
30    team1 = @teams_lookup[first_team.id][:team]
31    team2 = @teams_lookup[second_team.id][:team]
32    winner = higher_mean_team(first_team, second_team) ?
33      team1 : team2
34    AppComponent::Prediction.new(team1, team2, winner)
35  end
36
37  def higher_mean_team(first_team, second_team)
38    @teams_lookup[first_team.id][:rating].first.mean >
39      @teams_lookup[second_team.id][:rating].first.mean
40  end
41 end
42 end

```

```
$ bundle gem predictor --no-bin --no-coc --no-ext \
  --no-mit --test=rspec
$ mv predictor/ components/
$ cd components/predictor
$ rm -rf .git*
```

```
└── lib
    ├── predictor
    │   ├── prediction.rb
    │   └── predictor.rb
    └── spec
        ├── predictor_spec.rb
        └── spec_helper.rb
└── Gemfile
└── Gemfile.lock
└── README.md
└── Rakefile
└── predictor.gemspec
└── test.sh
```

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running predictor gem specs"
6 #bundle install | grep Installing
7 bundle exec rspec spec
8 exit_code+=$?
9
10 exit $exit_code
```

```
$ test.sh
*** Running predictor gem specs

. . .

sportsball/components/predictor/spec/predictor_spec.rb:1:in \
`<top (required)>': uninitialized constant AppComponent (NameError)
```

```
1 RSpec.describe AppComponent::Predictor do
2   before do
3     @team1 = create_team name: "A"
4     @team2 = create_team name: "B"
5
6     @predictor = AppComponent::Predictor.new([\@team1, \@team2])
7   end
8
9   it "predicts teams that have won in the past to win
10    in the future" do
11     game = create_game first_team: @team1, second_team: @team2,
12                         winning_team: 1
13     @predictor.learn([game])
14
15     prediction = @predictor.predict(@team2, @team1)
16     expect(prediction.winner).to eq @team1
17
18     prediction = @predictor.predict(@team1, @team2)
19     expect(prediction.winner).to eq @team1
20   end
21
```

```
22 it "changes predictions based on games learned" do
23   game1 = create_game first_team: @team1, second_team: @team2,
24     winning_team: 1
25   game2 = create_game first_team: @team1, second_team: @team2,
26     winning_team: 2
27   game3 = create_game first_team: @team1, second_team: @team2,
28     winning_team: 2
29   @predictor.learn([game1, game2, game3])
30
31   prediction = @predictor.predict(@team1, @team2)
32   expect(prediction.winner).to eq @team2
33 end
34
35 it "behaves funny when teams are equally strong" do
36   prediction = @predictor.predict(@team1, @team2)
37   expect(prediction).to be_an AppComponent::Prediction
38   expect(prediction.first_team).to eq @team1
39   expect(prediction.second_team).to eq @team2
40   expect(prediction.winner).to eq @team2
41
42   prediction = @predictor.predict(@team2, @team1)
43   expect(prediction).to be_an AppComponent::Prediction
44   expect(prediction.first_team).to eq @team2
45   expect(prediction.second_team).to eq @team1
46   expect(prediction.winner).to eq @team1
47 end
48 end
```

```
$ test.sh
*** Running predictor gem specs

. . .

sportsball/components/predictor/spec/predictor_spec.rb:1:in \
`<top (required)>': uninitialized constant Predictor::Predictor \
(NameError)
```

```

1 module Predictor
2   class Predictor
3     def initialize(teams)
4       @teams_lookup = teams.inject({}) do |memo, team|
5         memo[team.id] = {
6           team: team,
7
7           rating: [
8             Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
9           ]
10          }
11        memo
12      end
13    end
14
15
16    def learn(games)
17      games.each do |game|
18        first_team_rating = @teams_lookup[game.
19          first_team_id][:rating]
20        second_team_rating = @teams_lookup[
21          game.second_team_id][:rating]
22
23        game_result = game.winning_team == 1 ?
24          [first_team_rating, second_team_rating] :
25          [second_team_rating, first_team_rating]
26        Saulabs::TrueSkill::FactorGraph.
27          new(game_result, [1, 2]).update_skills
28      end
29    end
30
31
32    def predict(first_team, second_team)
33      team1 = @teams_lookup[first_team.id][:team]
34      team2 = @teams_lookup[second_team.id][:team]
35      winner = higher_mean_team(first_team, second_team) ?
36        team1 : team2
37      ::Predictor::Prediction.new(team1, team2, winner)
38    end
39
40    def higher_mean_team(first_team, second_team)
41      @teams_lookup[first_team.id][:rating].first.mean >
42        @teams_lookup[second_team.id][:rating].first.mean
43    end
44  end

```

```
1 module Predictor
2   class Prediction
3     attr_reader :first_team, :second_team, :winner
4
5     def initialize(first_team, second_team, winner)
6       @first_team = first_team
7       @second_team = second_team
8       @winner = winner
9     end
10    end
11  end
```

```
1 module Predictor
2   require "predictor/predictor"
3   require "predictor/prediction"
4 end
```

```
$ test.sh
*** Running predictor gem specs

Predictor::Predictor
  predicts teams that have won in the past to win in the future \
    (FAILED - 1)
  changes predictions based on games learned (FAILED - 2)
  behaves funny when teams are equally strong (FAILED - 3)
```

Failures:

```
1) Predictor::Predictor predicts teams that have won in the past \
   to win in the future
Failure/Error: @team1 = create_team name: "A"
NoMethodError:
  undefined method `create_team' for \
  #<RSpec::ExampleGroups::PredictorPredictor:0x007fed8581e9b0>
  # ./spec/predictor_spec.rb:3:in `block (2 levels) in
  # <top (required)>'

# ...
```

```
Finished in 0.00099 seconds (files took 0.69315 seconds to load)
3 examples, 3 failures
```

```
1 require_relative "spec_helper.rb"
2
3 RSpec.describe Predictor::Predictor do
4   before do
5     @team1 = OpenStruct.new(id: 6)
6     @team2 = OpenStruct.new(id: 7)
7
8     @predictor = Predictor::Predictor.new([\@team1, \@team2])
9   end
10
11  it "predicts teams that have won in the past to
12    win in the future" do
13    game = OpenStruct.new(
14      first_team_id: @team1.id, second_team_id: @team2.id,
15      winning_team: 1)
16    @predictor.learn([game])
17
18    prediction = @predictor.predict(@team2, @team1)
19    expect(prediction.winner).to eq @team1
20
21    prediction = @predictor.predict(@team1, @team2)
22    expect(prediction.winner).to eq @team1
23  end
24
25  it "changes predictions based on games learned" do
```

```
26   game1 = OpenStruct.new(first_team_id: @team1.id,
27     second_team_id: @team2.id, winning_team: 1)
28   game2 = OpenStruct.new(first_team_id: @team1.id,
29     second_team_id: @team2.id, winning_team: 2)
30   game3 = OpenStruct.new(first_team_id: @team1.id,
31     second_team_id: @team2.id, winning_team: 2)
32   @predictor.learn([game1, game2, game3])
33
34   prediction = @predictor.predict(@team1, @team2)
35   expect(prediction.winner).to eq @team2
36 end
37
38 it "behaves funny when teams are equally strong" do
39   prediction = @predictor.predict(@team1, @team2)
40   expect(prediction.first_team).to eq @team1
41   expect(prediction.second_team).to eq @team2
42   expect(prediction.winner).to eq @team2
43
44   prediction = @predictor.predict(@team2, @team1)
45   expect(prediction.first_team).to eq @team2
46   expect(prediction.second_team).to eq @team1
47   expect(prediction.winner).to eq @team1
48 end
49 end
```

```
1 require "bundler/setup"  
2 require "predictor"  
3 require "ostruct"
```

```
$ test.sh
*** Running predictor gem specs

Predictor::Predictor
predicts teams that have won in the past to win in the future \
(FAILED - 1)
changes predictions based on games learned (FAILED - 2)
behaves funny when teams are equally strong (FAILED - 3)

Failures:

1) Predictor::Predictor predicts teams that have won in the past \
   to win in the future
Failure/Error: @predictor = \
  Predictor::Predictor.new([@team1, @team2])
NameError:
  uninitialized constant Predictor::Predictor::Saulabs

...
Finished in 0.00102 seconds (files took 0.16532 seconds to load)
3 examples, 3 failures
```

```
1 # coding: utf-8
2 lib = File.expand_path("../lib", __FILE__)
3 $LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)
4
5 Gem::Specification.new do |spec|
6   spec.name          = "predictor"
7   spec.version       = "0.1.0"
8   spec.authors       = ["Stephan Hagemann"]
9   spec.email         = ["stephan.hagemann@gmail.com"]
10
11   spec.summary       = %q{Prediction Core}
12
13   spec.files = Dir["{lib}/**/*", "Rakefile", "README.md"]
14   spec.require_paths = ["lib"]
15
16   spec.add_dependency "trueskill"
17
18   spec.add_development_dependency "bundler"
19   spec.add_development_dependency "rake"
20   spec.add_development_dependency "rspec"
21 end
```

```
1 source "https://rubygems.org"
2
3 # Specify your gems dependencies in predictor.gemspec
4 gemspec
5
6 gem "trueskill",
7   git: "https://github.com/benjaminleesmith/trueskill",
8   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

```
1 require "saulabs/trueskill"
2
3 module Predictor
4   require "predictor/predictor"
5   require "predictor/prediction"
6 end
```

```
$ test.sh  
*** Running predictor gem specs
```

```
...
```

```
Finished in 0.00269 seconds (files took 0.12203 seconds to load)  
3 examples, 0 failures
```

```
$ test.sh
*** Running app component engine specs
..F.....F.....
```

Failures:

- 1) AppComponent::PredictionsController POST create assigns a prediction as @prediction  
Failure/Error: post :create,  
NameError:  
  uninitialized constant AppComponent::Predictor  
# ./app/controllers/app\_component/predictions\_controller.rb:8:  
#   :in `create'  
# ...
- 2) the prediction process get a new prediction  
Failure/Error: click\_button 'What is it going to be'  
NameError:  
  uninitialized constant AppComponent::Predictor  
# ./app/controllers/app\_component/predictions\_controller.rb:8:  
#   in `create'  
# ...

Finished in 1.64 seconds (files took 1.68 seconds to load)
69 examples, 2 failures

```
$ test.sh
*** Running app component engine specs
.....F.....F...
```

Failures:

- 1) AppComponent::PredictionsController POST create assigns a \  
 prediction as @prediction  
Failure/Error: post :create,  
NameError:  
    uninitialized constant AppComponent::PredictionsController::\  
        Predictor  
# ./app/controllers/app\_component/predictions\_controller.rb:8:\  
#      in `create'  
#...
- 2) the prediction process get a new prediction  
Failure/Error: click\_button 'What is it going to be'  
NameError:  
    uninitialized constant AppComponent::PredictionsController::\  
        Predictor  
# ./app/controllers/app\_component/predictions\_controller.rb:8:\  
#      in `create'  
#...

Finished in 1.96 seconds (files took 3.05 seconds to load)  
69 examples, 2 failures

```
1 $:.push File.expand_path("../lib", __FILE__ )
2
3 # Maintain your gems version:
4 require "app_component/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name          = "app_component"
9   s.version       = AppComponent::VERSION
10  s.authors       = ["Stephan Hagemann"]
11  s.email         = ["stephan.hagemann@gmail.com"]
12  s.homepage      = ""
13  s.summary        = "Summary of AppComponent."
14  s.description    = "Description of AppComponent."
15  s.license        = "MIT"
16
17  s.files = Dir["{app,config,db,lib}/**/*",
18                "MIT-LICENSE", "Rakefile", "README.md"]
19
20  s.add_dependency "rails", "5.1.4"
21  s.add_dependency "slim-rails", "3.1.3"
22  s.add_dependency "jquery-rails", "4.3.1"
23  s.add_dependency "trueskill"
24  s.add_dependency "predictor"
25
26  s.add_development_dependency "sqlite3"
27  s.add_development_dependency "rspec-rails"
28  s.add_development_dependency "shoulda-matchers"
29  s.add_development_dependency "database_cleaner"
30  s.add_development_dependency "capybara"
31  s.add_development_dependency "rails-controller-testing"
32 end
```

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 path "../" do
6   gem "predictor"
7 end
8
9 gem "trueskill",
10 git: "https://github.com/benjaminleesmith/trueskill",
11 ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

```
1 require "slim-rails"
2 require "jquery-rails"
3
4 module AppComponent
5   require "app_component/engine"
6   require "predictor"
7 end
```

Bundler could not find compatible versions for gem "predictor":

In Gemfile:

```
app_component (>= 0) ruby depends on  
  predictor (= 0.3.14159265359) ruby
```

Could not find gem 'predictor (= 0.3.14159265359) ruby',  
which is required by gem 'app\_component (>= 0) ruby',  
in any of the sources.

```
$ test.sh
*** Running app component engine specs
.....
Finished in 1.69 seconds (files took 1.79 seconds to load)
69 examples, 0 failures

Randomized with seed 35251
```

```
$ test.sh
*** Running container app specs
```

```
...
```

```
Randomized with seed 11686
```

```
....
```

```
Finished in 0.65533 seconds (files took 2.55 seconds to load)
4 examples, 0 failures
```

```
$ rails plugin new components/games_admin --full \
  --mountable --skip-bundle --skip-git --skip-test \
  --skip-system-test --dummy-path=spec/dummy
```

```
$ mkdir -p components/games_admin/app/views/games_admin
$ mkdir -p components/games_admin/spec/controllers/games_admin
$ mkdir -p components/games_admin/spec/features
```

```
$ mv components/app_component/app/controllers/app_component/\  
games_controller.rb \  
  components/games_admin/app/controllers/games_admin/\  
$ mv components/app_component/app/views/app_component/games/\  
  components/games_admin/app/views/games_admin/games  
  
$ mv components/app_component/spec/controllers/app_component/\  
games_controller_spec.rb\  
  components/games_admin/spec/controllers/games_admin/\  
$ mv components/app_component/spec/features/games_spec.rb\  
  components/games_admin/spec/features
```

```
1 $ sed -i -- 's/ Game\./ AppComponent::Game\./g' components/\ 
2 games_admin/\ 
3 app/controllers/games_admin/games_controller.rb 
4 
5 $ grep -rl "module AppComponent" components/games_admin/ | \ 
6 xargs sed -i '' 's/module AppComponent/module GamesAdmin/g' 
7 $ grep -rl "AppComponent::GamesController" components/games_admin/\ 
8 | xargs sed -i '' \ 
9   's/AppComponent::GamesController/GamesAdmin::GamesController/g' 
10 $ grep -rl "AppComponent::Engine" components/games_admin/ | \ 
11 xargs sed -i '' 's/AppComponent::Engine/GamesAdmin::Engine/g' 
12 $ grep -rl "app_component/" components/games_admin/ | \ 
13 xargs sed -i '' 's;app_component/;games_admin/;g' 
14 $ grep -rl "app_component\." components/games_admin/ | \ 
15 xargs sed -i '' 's;app_component\.;games_admin\.;g'
```

```
1 s.add_dependency "rails", "5.1.4"
2 s.add_dependency "slim-rails", "3.1.3"
3 s.add_dependency "jquery-rails", "4.3.1"
4 s.add_dependency "app_component"
5
6 s.add_development_dependency "sqlite3"
7 s.add_development_dependency "rspec-rails"
8 s.add_development_dependency "shoulda-matchers"
9 s.add_development_dependency "database_cleaner"
10 s.add_development_dependency "capybara"
11 s.add_development_dependency "rails-controller-testing"
```

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 path ".." do
6   gem "app_component"
7 end
8
9 gem "trueskill",
10   git: "https://github.com/benjaminleesmith/trueskill",
11   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

```
1 module GamesAdmin
2   class Engine < ::Rails::Engine
3     isolate_namespace GamesAdmin
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11
12    config.generators do |g|
13      g.orm           :active_record
14      g.template_engine :slim
15      g.test_framework :rspec
16    end
17  end
18 end
```

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10
11 require "rails-controller-testing"
12 Rails::Controller::Testing.install
13
14 Dir[GamesAdmin::Engine.root.join("spec/support/**/*.rb")].
15   each { |f| require f }
16
17 RSpec.configure do |config|
18   config.expect_with :rspec do |expectations|
19     expectations.
20       include_chain_clauses_in_custom_matcher_descriptions = true
21   end
22   config.mock_with :rspec do |mocks|
23     mocks.verify_partial_doubles = true
24   end
25
26   config.infer_spec_type_from_file_location!
27   config.disable_monkey_patching!
28   config.warnings = false
29   config.profile_examples = nil
30   config.order = :random
31   Kernel.srand config.seed
32
33   config.before(:suite) do
34     DatabaseCleaner.strategy = :transaction
35     DatabaseCleaner.clean_with(:truncation)
36   end
37
```

```
38 config.around(:each) do |example|
39   DatabaseCleaner.cleaning do
40     example.run
41   end
42 end
43
44 [:controller, :view, :request].each do |type|
45   config.include ::Rails::Controller::Testing::
46     TestProcess, :type => type
47   config.include ::Rails::Controller::Testing::
48     TemplateAssertions, :type => type
49   config.include ::Rails::Controller::Testing::
50     Integration, :type => type
51 end
52
53 config.include ObjectCreationMethods
54 end
55
56 Shoulda::Matchers.configure do |config|
57   config.integrate do |with|
58     with.test_framework :rspec
59     with.library :rails
60   end
61 end
```

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running games admin engine specs"
6 bundle install | grep Installing
7 bundle exec rake db:create db:migrate
8 RAILS_ENV=test bundle exec rake db:create
9 RAILS_ENV=test bundle exec rake db:migrate
10 bundle exec rspec spec
11 exit_code+=$?
12
13 exit $exit_code
```

```
$ test.sh  
.  
.  
.  
  
sportsball/components/games_admin/spec/spec_helper.rb:39:in  
`block in <top (required)>': uninitialized constant  
ObjectCreationMethods (NameError)
```

```
1 require_relative "../../spec/support/object_creation_methods.rb"
```

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10
11 require "rails-controller-testing"
12 Rails::Controller::Testing.install
13
14 Dir[GamesAdmin::Engine.root.join("spec/support/**/*.rb")].
15   each { |f| require f }
16
17 require "app_component/test_helpers"
18
19 RSpec.configure do |config|
20   config.expect_with :rspec do |expectations|
21     expectations.
22       include_chain_clauses_in_custom_matcher_descriptions = true
23   end
24   config.mock_with :rspec do |mocks|
25     mocks.verify_partial_doubles = true
26 end
```

```
27
28 config.infer_spec_type_from_file_location!
29 config.disable_monkey_patching!
30 config.warnings = false
31 config.profile_examples = nil
32 config.order = :random
33 Kernel.srand config.seed
34
35 config.before(:suite) do
36   DatabaseCleaner.strategy = :transaction
37   DatabaseCleaner.clean_with(:truncation)
38 end
39
40 config.around(:each) do |example|
41   DatabaseCleaner.cleaning do
42     example.run
43   end
44 end
45
46 [:controller, :view, :request].each do |type|
47   config.include ::Rails::Controller::Testing::
48     TestProcess, :type => type
49   config.include ::Rails::Controller::Testing::
50     TemplateAssertions, :type => type
51   config.include ::Rails::Controller::Testing::
52     Integration, :type => type
53 end
54
55 config.include AppComponent::ObjectCreationMethods
56 end
57
58 Shoulda::Matchers.configure do |config|
59   config.integrate do |with|
60     with.test_framework :rspec
61     with.library :rails
62   end
63 end
```

```
1 GamesAdmin::Engine.routes.draw do
2   resources :games
3 end
```

```
1 path "components" do
2   gem "app_component"
3   gem "games_admin"
4 end
```

```
1 Rails.application.routes.draw do
2   mount AppComponent::Engine, at: "/app_component"
3   mount GamesAdmin::Engine, at: "/games_admin"
4
5   root to: "app_component/welcome#show"
6 end
```

```
$ test.sh
Randomized with seed 34458
FFFF
```

#### Failures:

- 1) Engine 'App' has teams  
Failure/Error: visit "/"  
ActionView::Template::Error:  
undefined local variable or method `games\_path' for  
#<#<Class:0x007fcd19376188>:0x007fcd19374518>
- 2) Engine 'App' has games  
Failure/Error: visit "/"  
ActionView::Template::Error:  
undefined local variable or method `games\_path' for  
#<#<Class:0x007fcd19376188>:0x007fcd1e0c29a0>
- 3) Engine 'App' can predict  
Failure/Error: visit "/"  
ActionView::Template::Error:  
undefined local variable or method `games\_path' for  
#<#<Class:0x007fcd19376188>:0x007fcd19e9b590>
- 4) Engine 'App' hooks up to /  
Failure/Error: visit "/"  
ActionView::Template::Error:  
undefined local variable or method `games\_path' for  
#<#<Class:0x007fcd19376188>:0x007fcd19df89d0>

```
$ ack games_path components/app_component/app/views/layouts/\
app_component/application.html.slim
32: li =link_to "Games", games_path

components/games_admin/app/views/games_admin/games/edit.html.slim
7:= link_to 'Back', games_path

components/games_admin/app/views/games_admin/games/new.html.slim
5:= link_to 'Back', games_path

components/games_admin/app/views/games_admin/games/show.html.slim
28:= link_to "Back", games_path, class: "button"

components/games_admin/lib/games_admin.rb
11:   {name: "Games", link: -> {::GamesAdmin::Engine.\
routes.url_helpers.games_path}}
```

```
1 section.top-bar-section
2   ul.left
3     li =link_to "Teams", teams_path
4     li =link_to "Games", games_path
5     li =link_to "Predictions", new_prediction_path
```

```
1 section.top-bar-section
2   ul.left
3     li =link_to "Teams", teams_path
4     li =link_to "Games", "/games_admin/games"
5     li =link_to "Predictions", new_prediction_path
```

```
$ test.sh
Randomized with seed 33266
.F..
```

## Failures:

- 1) Engine 'App' has games
  - Failure/Error: within "main h1" do
  - Capybara::ElementNotFound:
    - Unable to find css "main h1"

```
rm -rf components/games_admin/app/views/layouts
```

```
1 module GamesAdmin
2   class ApplicationController < ActionController::Base
3     layout "app_component/application"
4   end
5 end
```

```
$ test.sh
Randomized with seed 42316
.F..
```

Failures:

- 1) Engine 'App' has games  
Failure/Error: click\_link "Games"  
ActionView::Template::Error:  
undefined local variable or method `teams\_path' for  
#<#<Class:0x007fc0bba84660>:0x007fc0bba7f4d0>

```
1 section.top-bar-section
2   ul.left
3     li =link_to "Teams", "/app_component/teams"
4     li =link_to "Games", "/games_admin/games"
5     li =link_to "Predictions",
6       "/app_component/prediction/new"
```

```
1 RSpec.describe "nav entry" do
2   it "points at the list of games" do
3     entry = GamesAdmin.nav_entry
4     expect(entry[:name]).to eq "Games"
5     expect(entry[:link].call).to eq "/games_admin/games"
6   end
7 end
```

```
1 require "slim-rails"
2 require "jquery-rails"
3
4 require "app_component"
5
6 module GamesAdmin
7   require "games_admin/engine"
8
9   def self.nav_entry
10    {
11      name: "Games",
12      link: -> {::GamesAdmin::Engine.routes.url_helpers.games_path}
13    }
14  end
15 end
```

```
1 Rails.application.config.main_nav =  
2   [  
3     TeamsAdmin.nav_entry,  
4     GamesAdmin.nav_entry,  
5     PredictionUi.nav_entry  
6   ]
```

```
1 - Rails.application.config.main_nav.each do |nav_entry|
2   li =link_to nav_entry[:name], nav_entry[:link].call
```

```
$ rails plugin new components/teams \
--skip-yarn --skip-git --skip-action-mailer --skip-action-cable \
--skip-puma --skip-sprockets --skip-spring --skip-listen \
--skip-coffee --skip-javascript --skip-turbolinks \
--skip-test --skip-system-test \
--dummy-path=spec/dummy --full --mountable
```

```
$ mkdir -p components/teams/app/models/teams
$ mkdir -p components/teams/spec/models/teams
$ mkdir -p components/teams/spec/support
```

```
$ mv components/app_component/app/models/app_component/team.rb \
  components/teams/app/models/teams/
$ mv components/app_component/spec/models/app_component/team_spec.rb \
  components/teams/spec/models/teams/
$ grep -rl "AppComponent" components/teams/ | \
  xargs sed -i '' 's/module AppComponent/module Teams/g'
```

```
1 #!/bin/bash
2
3 exit_code=0
4
5 echo "*** Running teams engine specs"
6 bundle install | grep Installing
7 bundle exec rake db:create db:migrate
8 RAILS_ENV=test bundle exec rake db:create
9 RAILS_ENV=test bundle exec rake db:migrate
10 bundle exec rspec spec
11 exit_code+=$?
12
13 exit $exit_code
```

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "ostruct"
9
10 Dir[Teams::Engine.root.join("spec/support/**/*.rb")].
11     each { |f| require f }
12
13 RSpec.configure do |config|
14   config.expect_with :rspec do |expectations|
15     expectations.
16       include_chain_clauses_in_custom_matcher_descriptions = true
17   end
18   config.mock_with :rspec do |mocks|
19     mocks.verify_partial_doubles = true
20   end
21
22 config.infer_spec_type_from_file_location!
23 config.disable_monkey_patching!
24 config.warnings = false
25 config.profile_examples = nil
26 config.order = :random
27 Kernel.srand config.seed
28
29 config.before(:suite) do
30   DatabaseCleaner.strategy = :transaction
31   DatabaseCleaner.clean_with(:truncation)
32 end
```

```
33
34 config.around(:each) do |example|
35   DatabaseCleaner.cleaning do
36     example.run
37   end
38 end
39
40 config.include Teams::ObjectCreationMethods
41 end
42
43 Shoulda::Matchers.configure do |config|
44   config.integrate do |with|
45     with.test_framework :rspec
46     with.library :rails
47   end
48 end
```

```
1 $:.push File.expand_path("../lib", __FILE__ )
2
3 # Maintain your gem's version:
4 require "teams/version"
5
6 # Describe your gem and declare its dependencies:
7 Gem::Specification.new do |s|
8   s.name          = "teams"
9   s.version       = Teams::VERSION
10  s.authors       = ["Stephan Hagemann"]
11  s.email         = ["stephan.hagemann@gmail.com"]
12  s.homepage      = ""
13  s.summary        = "Summary of Teams."
14  s.description    = "Description of Teams."
15
16  s.files = Dir["{app,config,db,lib}/**/*",
17                "MIT-LICENSE", "Rakefile", "README.md"]
18
19  s.add_dependency "activerecord", "5.1.0"
20
21  s.add_development_dependency "sqlite3"
22  s.add_development_dependency "rspec-rails"
23  s.add_development_dependency "shoulda-matchers"
24  s.add_development_dependency "database_cleaner"
25 end
```

```
1 source "https://rubygems.org"
2
3 gemspec
```

```
$ test.sh
*** Running teams engine specs
rake aborted!
LoadError: cannot load such file -- active_job/railtie
from ./components/teams/spec/dummy/config/application.rb:9
```

```
1 require_relative "boot"
2
3 # Pick the frameworks you want:
4 require "active_record/railtie"
5 # require "action_controller/railtie"
6 # require "action_view/railtie"
7 # require "action_mailer/railtie"
8 # require "active_job/railtie"
9 # require "action_cable/engine"
10 # require "rails/test_unit/railtie"
11 # require "sprockets/railtie"
12
13 Bundler.require(*Rails.groups)
14 require "teams"
15
16 module Dummy
17   class Application < Rails::Application
18     # Initialize configuration defaults for
19     # originally generated Rails version.
20     config.load_defaults 5.1
21
22     # Settings in config/environments/* take precedence over
23     # those specified here.
24     # Application configuration should go into files in
25     # config/initializers -- all .rb files in that directory
26     # are automatically loaded.
27   end
28 end
```

```
$ test.sh
*** Running teams engine specs

. . .
./sportsball/components/teams/spec/spec_helper.rb:37:
in `block in <top (required)>':
uninitialized constant Teams::ObjectCreationMethods (NameError)
```

```
1 module Teams::ObjectCreationMethods
2   def new_team(overrides = {})
3     defaults = {
4       name: "Some name #{counter}"
5     }
6     Teams::Team.new { |team| apply(team, defaults, overrides) }
7   end
8
9   def create_team(overrides = {})
10    new_team(overrides).tap(&:save!)
11  end
12
13 private
14
15 def counter
16   @counter ||= 0
17   @counter += 1
18 end
19
20 def apply(object, defaults, overrides)
21   options = defaults.merge(overrides)
22   options.each do |method, value_or_proc|
23     object.__send__(
24       "#{method}=",
25       value_or_proc.is_a?(Proc) ?
26         value_or_proc.call : value_or_proc)
27   end
28 end
29 end
```

```
1 require_relative "../../spec/support/object_creation_methods.rb"
```

```
1 module AppComponent::ObjectCreationMethods
2   def new_game(overrides = {})
3     defaults = {
4       first_team: -> { new_team },
5       second_team: -> { new_team },
6       winning_team: 2,
7       first_team_score: 2,
8       second_team_score: 3,
9       location: "Somewhere",
10      date: Date.today
11    }
12
13    AppComponent::Game.new do |game|
14      apply(game, defaults, overrides)
15    end
16  end
17
18  def create_game(overrides = {})
19    new_game(overrides).tap(&:save!)
20  end
21
22  private
23
24  def counter
25    @counter ||= 0
26    @counter += 1
27  end
28
29  def apply(object, defaults, overrides)
30    options = defaults.merge(overrides)
31    options.each do |method, value_or_proc|
32      object.__send__(
33        "#{method}=", 
34        value_or_proc.is_a?(Proc) ? 
35          value_or_proc.call : value_or_proc)
36    end
37  end
38 end
```

```
$ test.sh
*** Running teams engine specs

. . .

./sportsball/components/teams/spec/models/teams/team_spec.rb:1:
in `<top (required)>':
uninitialized constant AppComponent (NameError)
```

```
$ grep -rl "AppComponent::Team" components/teams/ | \  
xargs sed -i '' 's/AppComponent::Team/Teams::Team/g'
```

```
$ test.sh
*** Running teams engine specs
```

```
Randomized with seed 20999
```

```
F
```

```
Failures:
```

- 1) Teams::Team
  - Failure/Error: it { should validate\_presence\_of(:name) }
  - ActiveRecord::StatementInvalid:
    - Could not find table 'teams\_teams'

```
$ mkdir -p components/teams/db/migrate
$ mv components/app_component/db/migrate/\
*_create_app_component_teams.rb components/teams/db/migrate
```

```
1 class MoveTeamFromAppComponentToTeams < ActiveRecord::Migration[5.0]
2   def change
3     rename_table :app_component_teams, :teams_teams
4   end
5 end
```

```
$ test.sh
*** Running teams engine specs
== 20150110181102 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
 -> 0.0018s
== 20150110181102 CreateAppComponentTeams: migrated (0.0019s) =====

== 20150825215500 MoveTeamFromAppComponentToTeams: migrating =====
-- rename_table(:app_component_teams, :teams_teams)
 -> 0.0007s
== 20150825215500 MoveTeamFromAppComponentToTeams: migrated (0.0007s)

== 20150110181102 CreateAppComponentTeams: migrating =====
-- create_table(:app_component_teams)
 -> 0.0011s
== 20150110181102 CreateAppComponentTeams: migrated (0.0012s) =====

== 20150825215500 MoveTeamFromAppComponentToTeams: migrating =====
-- rename_table(:app_component_teams, :teams_teams)
 -> 0.0006s
== 20150825215500 MoveTeamFromAppComponentToTeams: migrated (0.0007s)
```

Randomized with seed 26777

.

Finished in 0.09969 seconds (files took 1.12 seconds to load)
1 example, 0 failures

Randomized with seed 26777

```
1 module Teams
2   class Engine < ::Rails::Engine
3     isolate_namespace Teams
4
5     initializer :append_migrations do |app|
6       unless app.root.to_s.match root.to_s+File::SEPARATOR
7         app.config.paths["db/migrate"].concat(
8           config.paths["db/migrate"].expanded)
9       end
10    end
11
12    config.generators do |g|
13      g.orm             :active_record
14      g.test_framework :rspec
15    end
16  end
17 end
```

```
$ test.sh
```

```
...
```

```
1) AppComponent::Game should belong to second_team
Failure/Error: it { should belong_to :second_team}
  Expected AppComponent::Game to have a belongs_to \
association called second_team (Teams::Team does not exist)
```

```
$ test.sh

. . .

1) games_admin/games/show renders attributes in <p>
Failure/Error: @game = assign(:game, create_game(\n
    location: "Location"))
NoMethodError:
  undefined method `new_team' for \
#<RSpec::ExampleGroups::GamesAdminGamesShow:0x007fca9c5a13d8>

2) games admin allows for the management of games
Failure/Error: team1 = create_team name: "UofL"
NoMethodError:
  undefined method `create_team' for \
#<RSpec::ExampleGroups::GamesAdmin:0x007fca9c573de8>
```

```
1 s.add_dependency "teams"
```

```
1 source "https://rubygems.org"
2
3 gemspec
4
5 path "../" do
6   gem "teams"
7 end
8
9 gem "trueskill",
10   git: "https://github.com/benjaminleesmith/trueskill",
11   ref: "e404f45af5b3fb86982881ce064a9c764cc6a901"
```

```
1 require "slim-rails"
2 require "jquery-rails"
3
4 require "teams"
5
6 module AppComponent
7   require "app_component/engine"
8 end
```

```
1 ENV["RAILS_ENV"] ||= "test"
2
3 require File.expand_path("../dummy/config/environment", __FILE__)
4
5 require "rspec/rails"
6 require "shoulda/matchers"
7 require "database_cleaner"
8 require "capybara/rails"
9 require "capybara/rspec"
10 require "ostruct"
11
12 require "rails-controller-testing"
13 Rails::Controller::Testing.install
14
15 Dir[AppComponent::Engine.root.join("spec/support/**/*.rb")].
16   each { |f| require f}
17
18 require "teams/test_helpers"
19
20 RSpec.configure do |config|
21   config.expect_with :rspec do |expectations|
22     expectations.
23       include_chain_clauses_in_custom_matcher_
24   descriptions = true
25 end
26 config.mock_with :rspec do |mocks|
27   mocks.verify_partial_doubles = true
28 end
29 config.infer_spec_type_from_file_location!
30 config.disable_monkey_patching!
31 config.warnings = false
32 config.profile_examples = nil
33 config.order = :random
34 Kernel.srand config.seed
35
36 config.before(:suite) do
37   DatabaseCleaner.strategy = :transaction
```

```
38     DatabaseCleaner.clean_with(:truncation)
39   end
40
41 config.around(:each) do |example|
42   DatabaseCleaner.cleaning do
43     example.run
44   end
45 end
46
47 [:controller, :view, :request].each do |type|
48   config.include ::Rails::Controller::Testing::
49     TestProcess, :type => type
50   config.include ::Rails::Controller::Testing::
51     TemplateAssertions, :type => type
52   config.include ::Rails::Controller::Testing::
53     Integration, :type => type
54 end
55
56 config.include Teams::ObjectCreationMethods
57 config.include AppComponent::ObjectCreationMethods
58 end
59
60 Shoulda::Matchers.configure do |config|
61   config.integrate do |with|
62     with.test_framework :rspec
63     with.library :rails
64   end
65 end
```

```
./components/app_component/
└── app
    ├── assets
    │   ├── images
    │   │   └── app_component
    │   ├── javascripts
    │   │   └── app_component
    │   └── stylesheets
    │       └── app_component
    ├── controllers
    │   └── app_component
    ├── models
    │   └── app_component
    └── views
        └── layouts
            └── app_component

    ├── bin
    ├── config
    │   └── initializers
    ├── lib
    │   └── app_component
    │       └── asks
    ├── spec
    │   ├── dummy
    │   ├── models
    │   │   └── app_component
    │   └── support
    └── vendor
        └── assets
            ├── javascripts
            │   └── app_component
            └── stylesheets
                └── app_component
```

```
$ find . -name \*app_component* -type f -print
./components/app_component/app/assets/config/\
    app_component_manifest.js
./components/app_component/app_component.gemspec
./components/app_component/lib/app_component.rb
./components/app_component/lib/tasks/app_component_tasks.rake
```

```
1 class RenameNamespaceOfGames < ActiveRecord::Migration
2   def change
3     rename_table :app_component_games, :web_ui_games
4   end
5 end
```

```
$ gem install cbratools
$ rnc AppComponent WebUi
  #renames the component
$ rnm AppComponent WebUi db/migrate db/schema.rb
  #creates needed migrations to support the component name change
```

```
1 #!/bin/bash --login
2
3 ensure() {
4     "$@" || exit 1
5 }
6
7 cd r4ia_examples/ticketee;
8
9 git checkout . && git clean -fd
10
11 rvm use 2.2.1@r4ia --create
12
13 gem install bundler
14
15 bundle
```

```
17 #####
18 ### CREATE PERSISTENCE COMPONENT AND MOVE CODE
19 #####
20
21 rails plugin new components/persistence \
22   --full \
23   --skip-bundle --skip-git \
24   --skip-test-unit \
25   --skip-action-view --skip-sprockets --skip-javascript \
26   --dummy-path=spec/dummy
27
28 mv app/models/* components/persistence/app/models
29 mkdir -p components/persistence/spec/models
30 mv spec/models/* components/persistence/spec/models
31
32 #pushd components/persistence && ensure eval rspec && popd
33
34 cp spec/spec_helper.rb components/persistence/spec/spec_helper.rb
35 cp spec/rails_helper.rb components/persistence/spec/rails_helper.rb
36
37 #pushd components/persistence && ensure eval rspec && popd
38
39 echo 'ENV["RAILS_ENV"] ||= "test"
40 require "spec_helper"
41 require File.expand_path("../dummy/config/environment", __FILE__)
42 require "rspec/rails"
43
44 Dir[Rails.root.join("spec/support/**/*.rb")].each { |f| require f }
45
46 ActiveRecord::Migration.maintain_test_schema!
47
48 RSpec.configure do |config|
49   config.fixture_path = "#{::Rails.root}/spec/fixtures"
50
51   config.use_transactional_fixtures = false
52
53   config.infer_spec_type_from_file_location!
54
55   config.include Warden::Test::Helpers, type: :feature
56   config.after(type: :feature) { Warden.test_reset! }
57   config.include Devise::TestHelpers, type: :controller
58 end
59 ' > components/persistence/spec/rails_helper.rb
60
61 #pushd components/persistence && ensure eval rspec && popd
```

```
./ticketee/components/persistence ./ticketee
./ticketee/components/persistence/spec/rails_helper.rb:4:in `require':
  cannot load such file --
./ticketee/components/persistence/config/environment (LoadError)
```

```
./ticketee/components/persistence/spec/dummy/db/schema.rb
  doesn't exist yet.
Run `rake db:migrate` to create it, then try again. If you do not
intend to use a database, you should instead alter
./ticketee/components/persistence/spec/dummy/config/application.rb
to limit the frameworks that will be loaded.
./ticketee/components/persistence/app/models/attachment.rb:4:
  in '<class:Attachment>': uninitialized constant
    Attachment::AttachmentUploader (NameError)
```

```
63 mkdir -p components/persistence/db/migrate
64 mv db/migrate/* components/persistence/db/migrate
65
66 #pushd components/persistence && \
67 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \
68 #  popd
```

```
70 mkdir -p components/persistence/app/uploaders
71 mv app/uploaders/* components/persistence/app/uploaders
72
73 #pushd components/persistence && \
74 #   rake db:migrate RAILS_ENV=test && ensure eval rspec && \
75 #   popd
```

```
~/.rvm/gems/ruby-2.2.1@r4ia/gems/activerecord-4.2.1/
  lib/active_record/dynamic_matchers.rb:26:in `method_missing':
undefined method `devise' for #<Class:0x007fe8104c13c8>
  (NoMethodError)
```

```
77 echo '
78 $:.push File.expand_path("../lib", __FILE__)
79
80 # Maintain your gems version:
81 require "persistence/version"
82
83 # Describe your gem and declare its dependencies:
84 Gem::Specification.new do |s|
85   s.name          = "persistence"
86   s.version       = Persistence::VERSION
87   s.authors       = ["TODO: Write your name"]
88   s.email         = ["TODO: Write your email address"]
89   s.homepage      = "TODO"
90   s.summary        = "TODO: Summary of Persistence."
91   s.description    = "TODO: Description of Persistence."
92   s.license        = "MIT"
93
94   s.files = Dir["{app,config,db,lib}/**/*",
95                 "MIT-LICENSE", "Rakefile", "README.rdoc"]
96
97   s.add_dependency "rails", "4.2.1"
98   s.add_dependency "devise", "3.4.1"
99
100  s.add_development_dependency "sqlite3"
101  s.add_development_dependency "rspec-rails", "3.2.1"
102 end
103 ' > components/persistence/persistence.gemspec
104
105 #pushd components/persistence && \
106 # rake db:migrate RAILS_ENV=test && ensure eval rspec && \
107 # popd
```

```
./ticketee/components/persistence/spec/rails_helper.rb:17:  
  in `block in <top (required)>': uninitialized  
  constant Warden (NameError)
```

```
109 echo '  
110 require "devise"  
111  
112 module Persistence  
113   require "persistence/engine"  
114 end  
115 ' > components/persistence/lib/persistence.rb  
116  
117 #pushd components/persistence && \  
118 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \  
119 #  popd
```

```
./ticketee/components/persistence/app/uploaders/  
attachment_uploader.rb:3: in `<top (required)>': uninitialized  
constant CarrierWave  (NameError)
```

```
121 echo '
122 $:.push File.expand_path("../lib", __FILE__)
123
124 # Maintain your gems version:
125 require "persistence/version"
126
127 # Describe your gem and declare its dependencies:
128 Gem::Specification.new do |s|
129   s.name          = "persistence"
130   s.version       = Persistence::VERSION
131   s.authors       = ["TODO: Write your name"]
132   s.email         = ["TODO: Write your email address"]
133   s.homepage      = "TODO"
134   s.summary        = "TODO: Summary of Persistence."
135   s.description    = "TODO: Description of Persistence."
136   s.license        = "MIT"
137
138   s.files = Dir["{app,config,db,lib}/**/*",
139                 "MIT-LICENSE", "Rakefile", "README.rdoc"]
140
141   s.add_dependency "rails", "4.2.1"
142   s.add_dependency "devise", "3.4.1"
143   s.add_dependency "carrierwave", "0.10.0"
144
145   s.add_development_dependency "sqlite3"
146   s.add_development_dependency "rspec-rails", "3.2.1"
147 end
148 ' > components/persistence/persistence.gemspec
149
150 echo '
151 require "devise"
152 require "carrierwave"
153
154 module Persistence
155   require "persistence/engine"
156 end
157 ' > components/persistence/lib/persistence.rb
158
159 #pushd components/persistence && \
160 # rake db:migrate RAILS_ENV=test && ensure eval rspec && \
161 # popd
```

```
~/.rvm/gems/ruby-2.2.1@r4ia/gems/activerecord-4.2.1/lib/active_record  
/dynamic_matchers.rb:26:in `method_missing':  
  undefined method `searcher' for #<Class:0x007ff634144358>  
(NoMethodError)
```

```
163 echo '
164 $:.push File.expand_path("../lib", __FILE__)
165
166 # Maintain your gem's version:
167 require "persistence/version"
168
169 # Describe your gem and declare its dependencies:
170 Gem::Specification.new do |s|
171   s.name          = "persistence"
172   s.version       = Persistence::VERSION
173   s.authors       = ["TODO: Write your name"]
174   s.email         = ["TODO: Write your email address"]
175   s.homepage      = "TODO"
176   s.summary        = "TODO: Summary of Persistence."
177   s.description    = "TODO: Description of Persistence."
178   s.license        = "MIT"
179
180   s.files = Dir["{app,config,db,lib}/**/*", "MIT-LICENSE",
181                 "Rakefile", "README.rdoc"]
182
183   s.add_dependency "rails", "4.2.1"
184   s.add_dependency "devise", "3.4.1"
185   s.add_dependency "carrierwave", "0.10.0"
186   s.add_dependency "searcher"
187
188   s.add_development_dependency "sqlite3"
189   s.add_development_dependency "rspec-rails", "3.2.1"
190 end
191 ' > components/persistence/persistence.gemspec
192
193 echo '
194 source "https://rubygems.org"
195
196 gemspec
197
198 gem "searcher", github: "radar/searcher",
199     ref: "c2975124e11677825481ced9539f16f0cb0640de"
200 ' > components/persistence/Gemfile
201
202 echo '
203 require "devise"
204 require "carrierwave"
205 require "searcher"
206
207 module Persistence
```

```
208   require "persistence/engine"
209 end
210 ' > components/persistence/lib/persistence.rb
211
212 #pushd components/persistence && bundle && \
213 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \
214 #  popd
```

```
~/.rvm/gems/ruby-2.2.1@r4ia/gems/activerecord-4.2.1/
  lib/active_record/dynamic_matchers.rb:26:in `method_missing':
  undefined method `devise' for
  #<Class:0x007fe8104c13c8> (NoMethodError)
```

```
213 mkdir -p components/persistence/config/initializers
214 mv config/initializers/devise.rb \
215     components/persistence/config/initializers
216
217 #pushd components/persistence && bundle && \
218 #  rake db:migrate RAILS_ENV=test && ensure eval rspec && \
219 #  popd
```

Finished in 23.55 seconds (files took 4.48 seconds to load)  
140 examples, 0 failures

SUCCESS

```
225 rm -rf components/persistence/app/assets
226 rm -rf components/persistence/app/controllers
227 rm -rf components/persistence/app/helpers
228 rm -rf components/persistence/app/mailers
229 rm -rf components/persistence/app/models/concerns
230 rm -rf components/persistence/app/models/.keep
231 rm -rf components/persistence/app/views
232 rm -rf components/persistence/lib/tasks
```

```
1 require_dependency "prediction_ui/application_controller"
2 module PredictionUi
3   class PredictionsController < ApplicationController
4     def new
5       @teams = Teams::Team.all
6     end
7
8     def create
9       predictor = Predictor::Predictor.new(Teams::Team.all)
10      predictor.learn(Games::Game.all)
11      @prediction = predictor.predict(
12        Teams::Team.find(params["first_team"]["id"]),
13        Teams::Team.find(params["second_team"]["id"]))
14    end
15  end
16 end
```

```
1 module Publisher
2   def add_subscriber(object)
3     @subscribers ||= []
4     @subscribers << object
5   end
6
7   def publish(message, *args)
8     return if @subscribers == [] || @subscribers == nil
9     @subscribers.each do |subscriber|
10       if subscriber.respond_to?(message)
11         subscriber.send(message, *args)
12       end
13     end
14   end
15 end
```

```
1 module PredictGame
2   class PredictGame
3     include Publisher
4
5     def initialize(teams, games)
6       @predictor = ::Predictor::Predictor.new(teams)
7       @predictor.learn(games)
8     end
9
10    def perform(team1_id, team2_id)
11      if @predictor.game_predictable?(team1_id, team2_id)
12        publish(
13          :prediction_succeeded,
14          @predictor.predict(team1_id, team2_id))
15      else
16        publish(
17          :prediction_failed,
18          @predictor.predict(team1_id, team2_id),
19          "Prediction can not be performed with
20            a team against itself")
21      end
22    end
23  end
24 end
```

```
1 module PredictionUi
2   class PredictionsController < ApplicationController
3     def new
4       @teams = TeamsStore::TeamRepository.new.get_all
5     end
6
7     def create
8       game_predictor = PredictGame::PredictGame.new(
9         TeamsStore::TeamRepository.new.get_all,
10        Games::Game.all)
11      game_predictor.add_subscriber(PredictionResponse.new(self))
12      game_predictor.perform(
13        TeamsStore::TeamRepository.new.get(
14          params["first_team"]["id"]),
15        TeamsStore::TeamRepository.new.get(
16          params["second_team"]["id"]))
17     end
18
19   class PredictionResponse < SimpleDelegator
20     def prediction_succeeded(prediction)
21       render locals: {prediction: prediction, message: nil}
22     end
23
24     def prediction_failed(prediction, error_message)
25       render locals: {
26         prediction: prediction, message: error_message
27       }
28     end
29   end
30 end
31 end
```

```
1 require_dependency "teams_admin/application_controller"
2
3 module TeamsAdmin
4   class TeamsController < ApplicationController
5     before_action :ensure_dependencies
6     before_action :set_team, only: [
7       :show, :edit, :update, :destroy]
8
9   def index
10    @teams = @team_repository.get_all
11  end
12
13  def new
14    @team = Teams::Team.new
15  end
16
17  def edit
18  end
19
20  def create
21    team = Teams::Team.new(team_params[:id],
22                           team_params[:name])
23    @team = @team_repository.create(team)
24
25    if @team.persisted?
26      redirect_to teams_teams_url,
27                  notice: "Team was successfully created."
28    else
29      render :new
30    end
31  end
32
33  def update
34    if @team_repository.update(@team.id, team_params[:name])
35      redirect_to teams_teams_url,
36                  notice: "Team was successfully updated."
37    else
38      render :edit
39    end
40  end
41
42  def destroy
43    @team_repository.delete(@team.id)
44    redirect_to teams_teams_url,
45                  notice: "Team was successfully destroyed."
46  end
```

```
47
48  private
49    def set_team
50      @team = @team_repository.get(params[:id])
51    end
52
53    def team_params
54      params.require(:teams_team).permit(:name)
55    end
56
57    def ensure_dependencies
58      @team_repository = TeamsStore::TeamRepository.new
59    end
60  end
61 end
```

```
1 module Teams
2   class Team
3     include ActiveRecord::Conversion
4     include ActiveRecord::Validations
5     include ActiveRecord::Naming
6
7     attr_reader :id, :name
8
9     def initialize(id=nil, name=nil)
10      @id = id
11      @name = name
12    end
13
14    def persisted?
15      @id != nil
16    end
17
18    def ==(other)
19      other.is_a?(Teams::Team) && @id == other.:
20    end
21
22    def new_record?
23      !persisted?
24    end
25  end
26 end
```

```
1 module TeamsStore
2   class TeamRepository
3     def get_all
4       TeamRecord.all.map do |team_record|
5         team_from_record(team_record)
6       end
7     end
8
9     def get(id)
10    team_record = TeamRecord.find_by_id(id)
11    team_from_record(team_record)
12  end
13
14   def create(team)
15     team_record = TeamRecord.create(name: team.name)
16     team_from_record(team_record)
17   end
18
19   def update(id, name)
20     TeamRecord.find_by_id(id).update(name: name)
21   end
22
23   def delete(id)
24     TeamRecord.delete(id)
25   end
26
27   private
28
29   class TeamRecord < ActiveRecord::Base
30     self.table_name = "teams_teams"
31
32     validates :name, presence: true
33   end
34   private_constant(:TeamRecord)
35
36   def team_from_record(team_record)
37     Teams::Team.new(team_record.id, team_record.name)
38   end
39 end
40 end
```

```
1 $:.push File.expand_path("../lib", __FILE__)
2
3 # Describe your gem and declare its dependencies:
4 Gem::Specification.new do |s|
5   s.name          = "teams_store"
6   s.version       = "0.0.1"
7   s.authors       = ["Your name"]
8   s.email         = ["Your email"]
9   s.summary        = "Summary of TeamsStore."
10  s.description    = "Description of TeamsStore."
11
12  s.files = Dir["{app,config,db,lib}/**/*",
13    "MIT-LICENSE", "Rakefile", "README.rdoc"]
14
15  s.add_dependency "activerecord", "5.1.4"
16  s.add_dependency "teams"
17
18  s.add_development_dependency "rails", "5.1.4"
19  s.add_development_dependency "sqlite3"
20  s.add_development_dependency "rspec-rails"
21  s.add_development_dependency "shoulda-matchers"
22  s.add_development_dependency "database_cleaner"
23 end
```

```
1 $:.push File.expand_path("../lib", __FILE__ )
2
3 # Describe your gem and declare its dependencies:
4 Gem::Specification.new do |s|
5   s.name          = "teams_store"
6   s.version       = "0.0.1"
7   s.authors       = ["Your name"]
8   s.email         = ["Your email"]
9   s.summary       = "Summary of TeamsStore."
10  s.description   = "Description of TeamsStore."
11
12  s.files = Dir["{app,config,db,lib}/**/*",
13    "MIT-LICENSE", "Rakefile", "README.rdoc"]
14
15  s.add_dependency "teams"
16
17  s.add_development_dependency "rspec"
18 end
```

```
1 module TeamsStore
2   class TeamRepository
3     def get_all
4       TeamsStore::Db.get.values
5     end
6
7     def get(key)
8       id = key.to_i
9       TeamsStore::Db.get[id]
10    end
11
12    def create(team)
13      return team if [nil, ""].include? team.name
14
15      id = TeamsStore::Db.get.keys.max &&
16          TeamsStore::Db.get.keys.max + 1 || 1
17      TeamsStore::Db.get[id] = Teams::Team.new(id, team.name)
18    end
19
20    def update(key, name)
21      id = key.to_i
22      return false if [nil, ""].include? name
23
24      TeamsStore::Db.get[id] = Teams::Team.new(id, name)
25      true
26    end
27
28    def delete(key)
29      id = key.to_i
30
31      TeamsStore::Db.get.delete(id)
32      id
33    end
34  end
35 end
```

```
1 module TeamsStore
2   module Db
3     def self.reset
4       $teams_db = {}
5     end
6
7     def self.get
8       $teams_db
9     end
10    end
11  end
```

```
1 #!/bin/bash
2
3 result=0
4
5 echo "### TESTING EVERYTHING WITH TEAMS DB"
6
7 rm components/teams_store
8 ln -s teams_store_db components/teams_store
9
10 for test_script in $(find . -name test.sh); do
11   pushd `dirname $test_script` > /dev/null
12   ./test.sh
13   ((result+=${?}))
14   popd > /dev/null
15 done
16
17 echo "### TESTING EVERYTHING WITH TEAMS IN MEM"
18
19 rm components/teams_store
20 ln -s teams_store_mem components/teams_store
21
22 for test_script in $(find . -name test.sh); do
23   pushd `dirname $test_script` > /dev/null
24   ./test.sh
25   ((result+=${?}))
26   popd > /dev/null
27 done
28
29 if [ $result -eq 0 ]; then
30   echo "SUCCESS"
31 else
32   echo "FAILURE"
33 fi
34
35 exit $result
```

```
1 module PredictionUi
2   class PredictionsController < ApplicationController
3     def new
4       @teams = Teams::Team.all
5     end
6
7     def create
8       @prediction = GamePredictionContext.call(
9         Teams::Team.all,
10        Games::Game.all,
11        Teams::Team.find(params["first_team"]["id"]),
12        Teams::Team.find(params["second_team"]["id"]))
13      end
14    end
15  end
16
17
18 class GamePredictionContext
19   def self.call(teams, games, first_team, second_team)
20     GamePredictionContext.new(
21       teams, games, first_team, second_team).call
22   end
23
24   def initialize(teams, games, first_team, second_team)
25     @contenders = teams
26     @contenders.each do |contender|
27       contender.extend Predictor::Role::Contender
28     end
29
30     @hpis = games
```

```
31 @hpis.each do |game|
32   game.extend Predictor::Role::HistoricalPerformanceIndicator
33 end
34
35 @predictor = first_team
36 @predictor.extend Predictor::Role::Contender
37 @predictor.extend Predictor::Role::Predictor
38
39 @second_team = second_team
40 @second_team.extend Predictor::Role::Contender
41 end
42
43 def call
44   @predictor.opponent = @second_team
45   @predictor.contenders = @contenders
46
47   @predictor.learn(@hpis)
48   @predictor.predict
49 end
50 end
```

```
1 module Predictor
2   module Role
3     module Contender
4       def self.extended(base)
5         base.rating = [
6           Saulabs::TrueSkill::Rating.new(1500.0, 1000.0, 1.0)
7         ]
8       end
9
10      def rating
11        @rating
12      end
13
14      def rating=(value)
15        @rating = value
16      end
17
18      def mean_of_rating
19        @rating.first.mean
20      end
21    end
22  end
23 end
```

```
1 module Predictor
2   module Role
3     module HistoricalPerformanceIndicator
4       def order_of_teams
5         result = [first_team_id, second_team_id]
6         result.reverse! if winning_team != 1
7         result
8     end
9   end
10 end
11 end
```

```
1 module Predictor
2   module Role
3     module Predictor
4       def contenders=(contenders)
5         @contenders_lookup =
6           contenders.inject({}) do |memo, contender|
7             memo[contender.id] = contender
8             memo
9           end
10        end
11
12      def opponent=(value)
13        @opponent = value
14      end
15
16      def game_predictable?
17        self != @opponent
18      end
19
20      def learn(games)
21        games.each do |game|
22          game_result = game.order_of_teams.map do |team_id|
23            @contenders_lookup[team_id].rating
24          end
25          Saulabs::TrueSkill::FactorGraph.new(
26            game_result, [1, 2]).update_skills
27        end
28      end
29
30      def predict
31        if game_predictable?
32          ::Predictor::Prediction.new(
33            self,
34            @opponent,
35            likely_winner)
36        else
```

```
37         ::Predictor::PredictionError.new(
38             self,
39             @opponent,
40             "Two contenders needed for prediction")
41     end
42 end
43
44 private
45
46 def likely_winner
47     team1 = @contenders_lookup[id]
48     team2 = @contenders_lookup[@opponent.id]
49
50     team1.mean_of_rating > team2.mean_of_rating ? team1 : team2
51 end
52 end
53 end
54 end
```

```
$ tree ./ -P *.gradle --prune -I test
./
├── app
│   └── build.gradle
├── build.gradle
└── components
    ├── accounts
    │   └── build.gradle
    ├── build.gradle
    ├── jdbc-support
    │   └── build.gradle
    ├── projects
    │   └── build.gradle
    ├── rest-support
    │   └── build.gradle
    ├── test-support
    │   └── build.gradle
    └── users
        └── build.gradle
└── settings.gradle
```

```
1 rootProject.name = "appcontinuum"  
2  
3 include "app"  
4  
5 include "components:accounts"  
6 include "components:jdbc-support"  
7 include "components:projects"  
8 include "components:rest-support"  
9 include "components:test-support"  
10 include "components:users"
```

```
1 buildscript {
2     ext.kotlin_version = '1.0.6'
3     repositories {
4         mavenCentral()
5         jcenter()
6     }
7     dependencies {
8         classpath "org.jetbrains.kotlin:kotlin-gradle-plugin" +
9             "$kotlin_version"
10    }
11 }
12
13 subprojects {
14     group "io.barinek.continuum"
15
16     apply plugin: 'kotlin'
17
18     defaultTasks "clean", "build"
19
20     repositories {
21         mavenCentral()
22         jcenter()
23     }
24
25     dependencies {
26         compile "org.jetbrains.kotlin:kotlin-stdlib:" +
27             "$kotlin_version"
```

```
28     compile "org.jetbrains.kotlinx:kotlinx-support-jdk8:" +
29             "0.3"
30     compile 'com.fasterxml.jackson.core:' +
31             'jackson-core:2.8.4'
32     compile 'com.fasterxml.jackson.core:' +
33             'jackson-databind:2.8.4'
34     compile 'com.fasterxml.jackson.core:' +
35             'jackson-annotations:2.8.4'
36
37     compile "com.fasterxml.jackson.datatype:" +
38             "jackson-datatype-jsr310:2.8.4"
39     compile "com.fasterxml.jackson.module:" +
40             "jackson-module-kotlin:2.8.4"
41
42     compile group: 'org.slf4j', name: 'slf4j-api',
43             version: '1.7.10'
44     compile group: 'org.slf4j', name: 'slf4j-simple',
45             version: '1.7.10'
46
47     testCompile 'junit:junit:4.11'
48     testCompile "org.jetbrains.kotlin:kotlin-test-junit:" +
49             "$kotlin_version"
50 }
51
52 sourceSets {
53     main.kotlin.srcDir "src/main/kotlin"
54     test.kotlin.srcDir "src/test/kotlin"
55
56     test.resources.srcDir "src/test/resources"
57 }
58 }
```

```
1 version "1.0-SNAPSHOT"
2
3 dependencies {
4     compile project(":components:jdbc-support")
5     compile project(":components:rest-support")
6     compile project(":components:test-support")
7     compile project(":components:users")
8 }
```

```
$ tree ./ -P \*.csproj --prune
./
├── Applications
│   ├── AllocationsServer
│   │   └── AllocationsServer.csproj
│   ├── BacklogServer
│   │   └── BacklogServer.csproj
│   ├── RegistrationServer
│   │   └── RegistrationServer.csproj
│   └── TimesheetsServer
│       └── TimesheetsServer.csproj
└── Components
    ├── Accounts
    │   └── Accounts.csproj
    ├── AccountsTest
    │   └── AccountsTest.csproj
    ├── Allocations
    │   └── Allocations.csproj
    ├── AllocationsTest
    │   └── AllocationsTest.csproj
    ├── Backlog
    │   └── Backlog.csproj
    ├── BacklogTest
    │   └── BacklogTest.csproj
    ├── DatabaseSupport
    │   └── DatabaseSupport.csproj
    ├── DatabaseSupportTest
    │   └── DatabaseSupportTest.csproj
    ├── Projects
    │   └── Projects.csproj
    ├── ProjectsTest
    │   └── ProjectsTest.csproj
    ├── TestSupport
    │   └── TestSupport.csproj
    ├── Timesheets
    │   └── Timesheets.csproj
    ├── TimesheetsTest
    │   └── TimesheetsTest.csproj
    ├── Users
    │   └── Users.csproj
    └── UsersTest
        └── UsersTest.csproj
```

```
1 <Project Sdk="Microsoft.NET.Sdk">
2
3   <PropertyGroup>
4     <TargetFramework>netstandard1.6</TargetFramework>
5   </PropertyGroup>
6
7   <ItemGroup>
8     <ProjectReference
9       Include="..../DatabaseSupport/DatabaseSupport.csproj" />
10    <ProjectReference Include="..../Users/Users.csproj" />
11    <PackageReference Include=" MySqlConnector" Version="0.19.3" />
12  </ItemGroup>
13
14  <ItemGroup>
15    <PackageReference Include=" Microsoft.AspNetCore"
16      Version="1.1.2" />
17    <PackageReference Include=" Microsoft.AspNetCore.Mvc"
18      Version="1.1.3" />
19    <PackageReference Include=" Microsoft.Extensions.Logging.Debug"
20      Version="1.1.2" />
21  </ItemGroup>
22
23 </Project>
```

```
$ bundle gem bundle_gem
$ rails plugin new plain_plugin
$ rails plugin new full_engine --full
$ rails plugin new mountable_engine --mountable
$ rails plugin new full_mountable_engine --full --mountable
```

```
bundle_gem
└── bin
└── lib
    └── bundle_gem
└── spec
```

# plain\_plugin

```
├── bin
└── lib
    ├── plain_plugin
    └── tasks
        └── test
            └── dummy
                ├── app
                │   ├── assets
                │   │   ├── config
                │   │   ├── images
                │   │   ├── javascripts
                │   │   │   └── channels
                │   │   └── stylesheets
                │   ├── channels
                │   │   └── application_cable
                │   ├── controllers
                │   │   └── concerns
                │   ├── helpers
                │   ├── jobs
                │   ├── mailers
                │   ├── models
                │   │   └── concerns
                │   └── views
                │       └── layouts
                ├── bin
                ├── config
                │   ├── environments
                │   ├── initializers
                │   └── locales
                ├── db
                ├── lib
                │   └── assets
                ├── log
                ├── public
                └── tmp
                    └── cache
                        └── assets
```

```
1 require_relative 'boot'  
2  
3 require 'rails/all'  
4  
5 Bundler.require(*Rails.groups)  
6 require "plain_plugin"
```

```
1 require File.expand_path("../..../test/dummy/config/environment.rb",
2     __FILE__ )
3 ActiveRecord::Migrator.migrations_paths = [
4     File.expand_path("../..../test/dummy/db/migrate", __FILE__ )
5 ]
6 require "rails/test_help"
7
8 # Filter out Minitest backtrace while allowing backtrace
9 # from other libraries to be shown.
10 Minitest.backtrace_filter = Minitest::BacktraceFilter.new
```



```
full_engine
└── app
    ├── assets
    │   ├── config
    │   └── images
    │       └── full_engine
    ├── javascripts
    │   └── full_engine
    ├── stylesheets
    │   └── full_engine
    ├── controllers
    ├── helpers
    ├── mailers
    ├── models
    └── views
└── bin
└── config
└── lib
    └── full_engine
        └── tasks
└── test
    └── dummy
```

```
Error: Command not recognized
Usage: rails COMMAND [ARGS]
```

The common Rails commands available **for** engines are:

```
generate Generate new code (short-cut alias: "g")
destroy Undo code generated with "generate" (short-cut alias: "d")
```

All commands can be run **with -h for** more information.

If you want to run any commands that need to be run **in** context of the application, like `rails server` or `rails console`, you should **do** it from the application's directory (typically test/dummy).

spec/dummy

└── Rakefile

└── bin

└── rails

└── config

└── application.rb

└── boot.rb

└── database.yml

└── environment.rb

└── environments

└── development.rb

└── test.rb

└── routes.rb

└── db

└── schema.rb

└── log

└── .keep

```
1 module FullEngine
2   class Engine < ::Rails::Engine
3   end
4 end
```

```
1 Rails.application.routes.draw do
2   end
```



```
mountable_engine
└── app
    ├── assets
    │   ├── config
    │   ├── images
    │   │   └── mountable_engine
    │   ├── javascripts
    │   │   └── mountable_engine
    │   └── stylesheets
    │       └── mountable_engine
    ├── controllers
    │   └── mountable_engine
    ├── helpers
    │   └── mountable_engine
    ├── jobs
    │   └── mountable_engine
    ├── mailers
    │   └── mountable_engine
    ├── models
    │   └── mountable_engine
    └── views
        └── layouts
            └── mountable_engine
└── bin
└── config
└── lib
    └── mountable_engine
        └── tasks
└── test
    └── dummy
```

```
1 module MountableEngine
2   class Engine < ::Rails::Engine
3     isolate_namespace MountableEngine
4   end
5 end
```

```
1 module MountableEngine
2   class User < ApplicationRecord
3   end
4 end
```

```
1 MountableEngine::Engine.routes.draw do
2   resources :users
3 end
```

```
1 Rails.application.routes.draw do
2   mount MountableEngine::Engine => "/mountable_engine"
3 end
```

Prefix	Verb	URI Pattern	Controller#Action
mountable_engine		/mountable_engine	MountableEngine::Engin

Routes for MountableEngine::Engine:

users	GET	/users(:format)	moun...ngine/users#index
	POST	/users(:format)	moun...ngine/users#create
new_user	GET	/users/new(:format)	moun...ngine/users#new
edit_user	GET	/users/:id/edit(:format)	moun...ngine/users#edit
user	GET	/users/:id(:format)	moun...ngine/users#show
	PATCH	/users/:id(:format)	moun...ngine/users#update
	PUT	/users/:id(:format)	moun...ngine/users#update
	DELETE	/users/:id(:format)	moun...ngine/users#destroy

```
rake app:about
# List versions of all Rails frame
works and the environment
rake app:app:template
# Applies the template supplied by LOCATION=(/path/to/
    template)
# or URL
rake app:app:update
# Update configs and some other initially generated files
# (or use just update:configs or update:bin)
rake app:assets:clean[keep]
# Remove old compiled assets
rake app:assets:clobber
# Remove compiled assets
rake app:assets:environment
# Load asset compile environment
rake app:assets:precompile
# Compile all the assets named in config.assets.precompile
rake app:cache_digests:dependencies
# Lookup first-level dependencies for TEMPLATE
# (like messages/show or comments/_comment.html)
rake app:cache_digests:nested_dependencies
# Lookup nested dependencies for TEMPLATE
# (like messages/show or comments/_comment.html)
```

```
rake app:db:create
# Creates the database from DATABASE_URL or config/
# database.yml
# for the current RAILS_ENV (use db:create:all to create all
# databases in the config). Without RAILS_ENV or when
# RAILS_ENV
# is development, it defaults to creating the development and
# test databases
rake app:db:drop
# Drops the database from DATABASE_URL or config/database.yml
# for the current RAILS_ENV (use db:drop:all to drop all
# databases
# in the config). Without RAILS_ENV or when RAILS_ENV is
# development, it defaults to dropping the development and
# test databases
rake app:db:environment:set
# Set the environment value for the database
rake app:db:fixtures:load
# Loads fixtures into the current environment's database
rake app:db:migrate
# Migrate the database
# (options: VERSION=x, VERBOSE=false, SCOPE=blog)
rake app:db:migrate:status
# Display status of migrations
rake app:db:rollback
# Rolls the schema back to the previous version
# (specify steps w/ STEP=n)
rake app:db:schema:cache:clear
# Clears a db/schema_cache.yml file
rake app:db:schema:cache:dump
# Creates a db/schema_cache.yml file
rake app:db:schema:dump
# Creates a db/schema.rb file that is portable against
# any DB supported by Active Record
rake app:db:schema:load
# Loads a schema.rb file into the database
rake app:db:seed
# Loads the seed data from db/seeds.rb
rake app:db:setup
# Creates the database, loads the schema, and initializes
# with the seed data (use db:reset to also drop the database
# first)
rake app:db:structure:dump
# Dumps the database structure to db/structure.sql
rake app:db:structure:load
# Recreates the databases from the structure.sql file
```

```
rake app:db:version
  # Retrieves the current schema version number
rake app:dev:cache
  # Toggle development mode caching on/off
rake app:initializers
  # Print out all defined initializers in the order they are
  # invoked by Rails
rake app:log:clear
  # Truncates all/specified *.log files in log/ to zero bytes
  # (specify which logs with LOGS=test,development)
rake app:middleware
  # Prints out your Rack middleware stack
rake app:mountable_engine:install:migrations
  # Copy migrations from mountable_engine to application
rake app:notes
  # Enumerate all annotations
  # (use notes:optimize, :fixme, :todo for focus)
rake app:notes:custom
  # Enumerate a custom annotation, specify with ANNOTATION=CUSTOM
rake app:restart
  # Restart app by touching tmp/restart.txt
rake app:routes
  # Print out all defined routes in match order, with names
rake app:secret
  # Generate a cryptographically secure secret key (this is
  # typically used to generate a secret for cookie sessions)
rake app:test
  # Runs all tests in test folder except system ones
rake app:test:db
  # Run tests quickly, but also reset db
rake app:test:system
  # Run system tests only
rake app:time:zones[country_or_offset]
  # List all time zones, list by two-letter country code
  # (`rails time:zones[US]`), or list by UTC offset (`rails
  # time:zones[-8]`)
rake app:tmp:clear
  # Clear cache and socket files from tmp/
  # (narrow w/ tmp:cache:clear, tmp:sockets:clear)
rake app:tmp:create
  # Creates tmp directories for cache, sockets, and pids
rake app:update
  # Update some initially generated files
rake app:yarn:install
  # Install all JavaScript dependencies as specified via Yarn
```

```
rake build
# Build mountable_engine-0.1.0.gem into the pkg directory
rake clean
# Remove any temporary products
rake clobber
# Remove any generated files
rake clobber_rdoc
# Remove RDoc HTML files
rake db:create
# Create the database from config/database.yml for the
# current
# Rails.env
# (use db:create:all to create all databases in the config)
rake db:drop
# Drops the database for the current Rails.env
# (use db:drop:all to drop all databases)
rake db:fixtures:load
# Load fixtures into the current environment's database
rake db:migrate
# Migrate the database (options: VERSION=x, VERBOSE=false)
rake db:migrate:status
# Display status of migrations
```

```
rake db:rollback
  # Rolls the schema back to the previous version
  # (specify steps w/ STEP=n)
rake db:schema:dump
  # Create a db/schema.rb file that can be portably
  # used against any DB supported by Active Record
rake db:schema:load
  # Load a schema.rb file into the database
rake db:seed
  # Load the seed data from db/seeds.rb
rake db:setup
  # Create the database, load the schema, and initialize
  # with the seed data (use db:reset to also drop the db first)
rake db:structure:dump
  # Dump the database structure to an SQL file
rake db:version
  # Retrieves the current schema version number
rake install
  # Build and install mountable_engine-0.1.0.gem into system gems
rake install:local
  # Build and install mountable_engine-0.1.0.gem into system
  # gems
  # without network access
rake rdoc
  # Build RDoc HTML files
rake release[remote]
  # Create tag v0.1.0 and build and push mountable_engine-
  # 0.1.0.gem    # to Rubygems
rake rerdoc
  # Rebuild RDoc HTML files
rake stats
  # Report code statistics (KLOCs, etc) from the application
  # or engine
rake test
  # Run tests
```

```
1 Rails.application.routes.draw do
2   constraints subdomain: "sportsball" do
3     mount AppComponent::Engine, at: "/"
4   end
5 end
```

```
1 # For rails dev, so we can use subdomains
2 127.0.0.1 dev.localhost
3 127.0.0.1 sportsball.dev.localhost
```

```
1  mount Admin::Engine, at: "admin1", as: "admin1"
2  mount Admin::Engine, at: "admin2", as: "admin2"
```

```
1 RSpec.describe "app_component/teams/new", :type => :view do
2   before :each do
3     assign :team, AppComponent::Team.new(
4       :name => "MyString"
5     )
6   end
7
8   it "renders new team form" do
9     render
10    assert_select "form[action=?][method=?]",
11      app_component.teams_path, "post" do
12      assert_select "input#team_name[name=?]", "team[name]"
13    end
14  end
15 end
```

```
1 ActionView::TestCase::TestController.instance_eval do
2   helper AppComponent::Engine.routes.url_helpers
3 end
4 ActionView::TestCase::TestController.class_eval do
5   def _routes
6     AppComponent::Engine.routes
7   end
8 end
```

```
1 RSpec.describe AppComponent::TeamsController, :type => :routing do
2   routes { AppComponent::Engine.routes }
3
4   it "routes to #index" do
5     expect(:get => "/teams").to
6       route_to("app_component/teams#index")
7   end
8
9   it "routes to #new" do
10    expect(:get => "/teams/new").to route_to(
11      "app_component/teams#new")
12  end
13
14  it "routes to #show" do
15    expect(:get => "/teams/1").to
16      route_to("app_component/teams#show", id: "1")
17  end
18
19  it "routes to #edit" do
20    expect(:get => "/teams/1/edit").to
21      route_to("app_component/teams#edit", id: "1")
22  end
23
24  it "routes to #create" do
25    expect(:post => "/teams").to
26      route_to("app_component/teams#create")
27  end
28
29  it "routes to #update" do
30    expect(:put => "/teams/1").to
31      route_to("app_component/teams#update", id: "1")
32  end
33
34  it "routes to #destroy" do
35    expect(:delete => "/teams/1").to
36      route_to("app_component/teams#destroy", id: "1")
37  end
38 end
```