

Here are some figures to consider while answering the following questions.

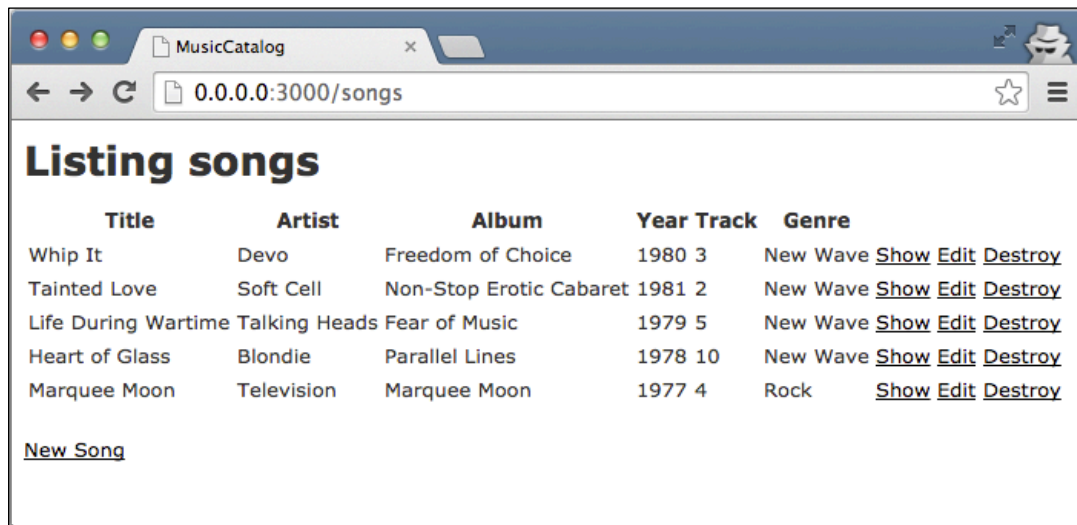


Figure 1. Example page from Music Catalog web app.

```
1 MusicCatalog::Application.routes.draw do
2   resources :songs
3 end
```

Figure 2. config/routes.rb

```
$ rake routes
Prefix Verb  URI Pattern          Controller#Action
songs  GET    /songs(.:format)    songs#index
       POST   /songs(.:format)    songs#create
new_song GET    /songs/new(.:format) songs#new
edit_song GET    /songs/:id/edit(.:format) songs#edit
song  GET    /songs/:id(.:format) songs#show
       PATCH  /songs/:id(.:format) songs#update
       PUT    /songs/:id(.:format) songs#update
       DELETE /songs/:id(.:format) songs#destroy
```

Figure 3. Output of rake routes command.

```
1 # == Schema Information
2 #
3 # Table name: songs
4 #
5 # id          :integer          not null, primary key
6 # title       :string(255)
7 # artist      :string(255)
8 # album       :string(255)
9 # year        :string(255)
10 # track       :integer
11 # genre       :string(255)
12 # created_at  :datetime
13 # updated_at  :datetime
14 #
15
16 class Song < ActiveRecord::Base
17 end
```

Figure 4. app/models/song.rb

```
1 class CreateSongs < ActiveRecord::Migration
2   def change
3     create_table :songs do |t|
4       t.string :title
5       t.string :artist
6       t.string :album
7       t.string :year
8       t.integer :track
9       t.string :genre
10
11       t.timestamps
12     end
13   end
14 end
```

Figure 5. db/migrate/20140930033607_create_songs.rb

```

1 class SongsController < ApplicationController
2   def index
3     @songs = Song.all
4   end
5
6   def show
7     @song = Song.find(params[:id])
8   end
9
10  def new
11    @song = Song.new
12  end
13
14  def edit
15    @song = Song.find(params[:id])
16  end
17
18  def create
19    @song = Song.new(song_params)
20    respond_to do |format|
21      if @song.save
22        format.html { redirect_to @song, notice: 'Song was successfully created.' }
23        format.json { render action: 'show', status: :created, location: @song }
24      else
25        format.html { render action: 'new' }
26        format.json { render json: @song.errors, status: :unprocessable_entity }
27      end
28    end
29  end
30
31  def update
32    @song = Song.find(params[:id])
33    respond_to do |format|
34      if @song.update(song_params)
35        format.html { redirect_to @song, notice: 'Song was successfully updated.' }
36        format.json { head :no_content }
37      else
38        format.html { render action: 'edit' }
39        format.json { render json: @song.errors, status: :unprocessable_entity }
40      end
41    end
42  end
43
44  def destroy
45    @song = Song.find(params[:id])
46    @song.destroy
47    respond_to do |format|
48      format.html { redirect_to songs_url }
49      format.json { head :no_content }
50    end
51  end
52
53  private
54  # Never trust parameters from the scary internet, only allow the white list through.
55  def song_params
56    params.require(:song).permit(:title, :artist, :album, :year, :track, :genre)
57  end
58 end

```

Figure 6. app/controllers/songs_controller.rb

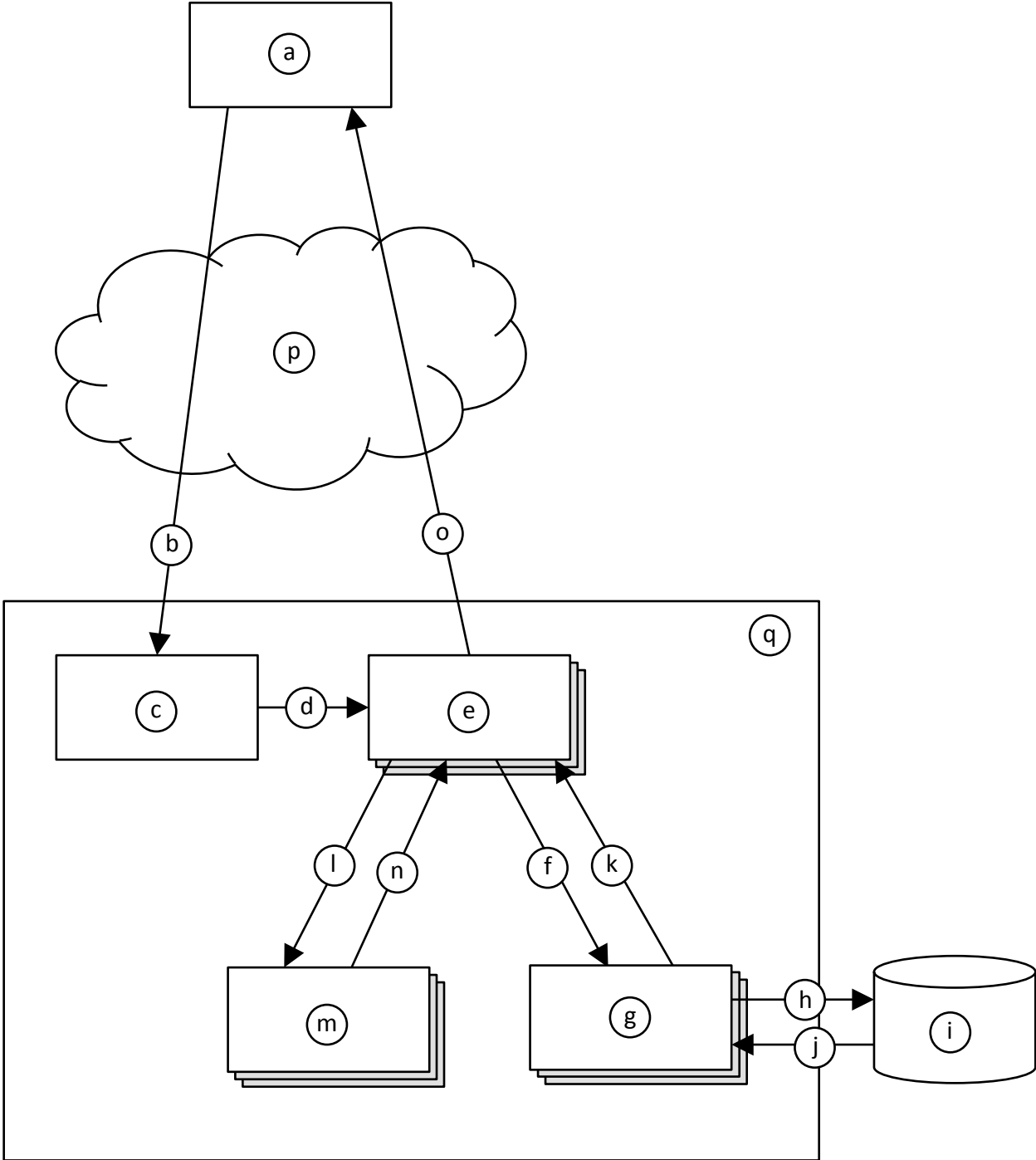
```

1 <h1>Listing songs</h1>
2
3 <table>
4   <thead>
5     <tr>
6       <th>Title</th>
7       <th>Artist</th>
8       <th>Album</th>
9       <th>Year</th>
10      <th>Track</th>
11      <th>Genre</th>
12      <th></th>
13      <th></th>
14      <th></th>
15    </tr>
16  </thead>
17
18  <tbody>
19    <% @songs.each do |song| %>
20      <tr>
21        <td><%= song.title %></td>
22        <td><%= song.artist %></td>
23        <td><%= song.album %></td>
24        <td><%= song.year %></td>
25        <td><%= song.track %></td>
26        <td><%= song.genre %></td>
27        <td><%= link_to 'Show', song %></td>
28        <td><%= link_to 'Edit', edit_song_path(song) %></td>
29        <td><%= link_to 'Destroy', song, method: :delete, data: { confirm: 'Are you sure?' } %></td>
30      </tr>
31    <% end %>
32  </tbody>
33 </table>
34
35 <br>
36
37 <%= link_to 'New Song', new_song_path %>

```

Figure 7. app/views/songs/index.html.erb

Problem: First consider this figure depicting the Rails MVC architecture.



Now, given the architectural diagram, think about how the web page in Figure 1 would have come to be displayed. Fill in each lettered item from the figure (blanks at left) the most appropriate label number (at right). Note that you will not use all of the label numbers.

- | | |
|----------|--|
| a. _____ | 1) routes.rb (Figure 2) |
| b. _____ | 2) song.rb (Figure 4) |
| c. _____ | 3) 20140930033607_create_songs.rb (Figure 5) |
| d. _____ | 4) songs_controller.rb (Figure 6) |
| e. _____ | 5) index.html.erb (Figure 7) |
| f. _____ | 6) Ye Olde Internet |
| g. _____ | 7) Rails server |
| h. _____ | 8) Web browser |
| i. _____ | 9) Call to SongsController#index |
| j. _____ | 10) Call to SongsController#show |
| k. _____ | 11) Call to Song::all |
| l. _____ | 12) Data returned by Song::all |
| m. _____ | 13) Call to Song::find |
| n. _____ | 14) Data returned by Song::find |
| o. _____ | 15) Call to CreateSongs#change |
| p. _____ | 16) Data returned from CreateSongs#change |
| q. _____ | 17) Call to index.html.erb (whatever that means) |
| | 18) Data returned from index.html.erb |
| | 19) Invocation of SQL query |
| | 20) Data returned form SQL query |
| | 21) HTTP GET request |
| | 22) HTTP response |
| | 23) Database |

Solution:

a. 8

b. 21

c. 1

d. 9

e. 4

f. 11

g. 2

h. 19

i. 23

j. 20

k. 12

l. 17

m. 5

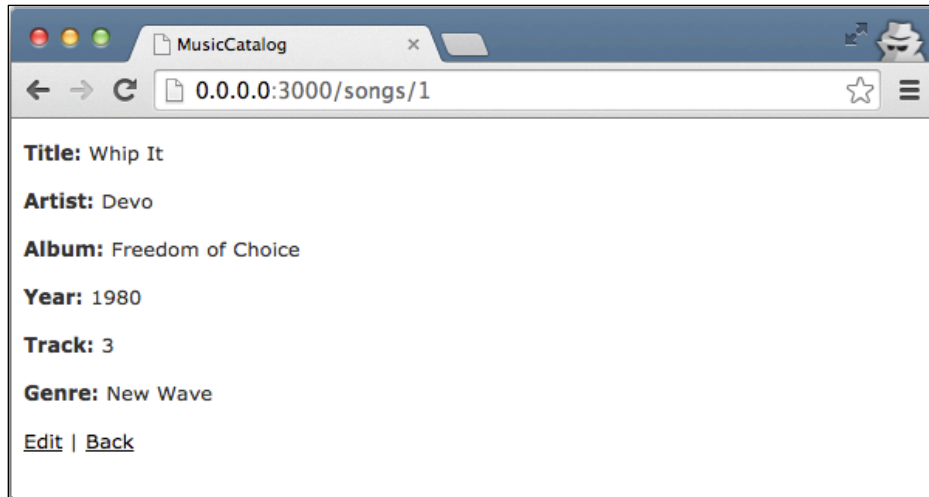
n. 18

o. 22

p. 6

q. 7

Problem: In Figure 1, if you were to click the “Show” link for “Whip It”, this page would display.



Write the ERB file for this page. Assume that a layout, `application.html.erb`, already exists, so your ERB need only include the main content being displayed. Your ERB must include the following types of HTML elements: **p** and **strong**.

Solution:

It's OK to omit line 1.

```
1 <p id="notice"><%= notice %></p>
2
3 <p>
4   <strong>Title:</strong>
5   <%= @song.title %>
6 </p>
7
8 <p>
9   <strong>Artist:</strong>
10  <%= @song.artist %>
11 </p>
12
13 <p>
14   <strong>Album:</strong>
15   <%= @song.album %>
16 </p>
17
18 <p>
19   <strong>Year:</strong>
20   <%= @song.year %>
21 </p>
22
23 <p>
24   <strong>Track:</strong>
25   <%= @song.track %>
26 </p>
27
28 <p>
29   <strong>Genre:</strong>
30   <%= @song.genre %>
31 </p>
32
33 <%= link_to 'Edit', edit_song_path(@song) %> |
34 <%= link_to 'Back', songs_path %>
```

Problem: Modify the web app such that the page from Figure 1 includes only songs from 1980 or later. Here are a few hints:

- To create a new array:
 - `my_array = Array.new`
- To add an item to the end of an array:
 - `my_array.push(my_item)`
- To convert a string to an integer:
 - `my_int = my_string.to_i`

Solution:

Here's one straightforward way to solve the problem by changing SongsController#index (in songs_controller.rb):

```
1  class SongsController < ApplicationController
2    def index
3      # BEFORE:
4      #@songs = Song.all
5      #
6      # AFTER:
7      @songs = Array.new
8      Song.all.each do |song|
9        if song.year.to_i >= 1980 then
10         @songs.push(song)
11       end
12     end
13   end
```

(The rest of the file remains unchanged.)

Problem: Imagine that you wanted to change the web app such that it now stores the name of the songwriter with each song. Answer the following in plain English.

- a. How would you go about updating the web app's "M" (as in MVC) component?
- b. How would you change the "V" files in the above figures?
- c. How would you change the "C" files in the above figures?

Solution:

- a. To update the model (“M”) component, you would need to create a new migration (similar to Figure 5). A common way to do this would be with this Rails command:

```
$ rails generate migration AddSongwriterToSongs songwriter:string
```

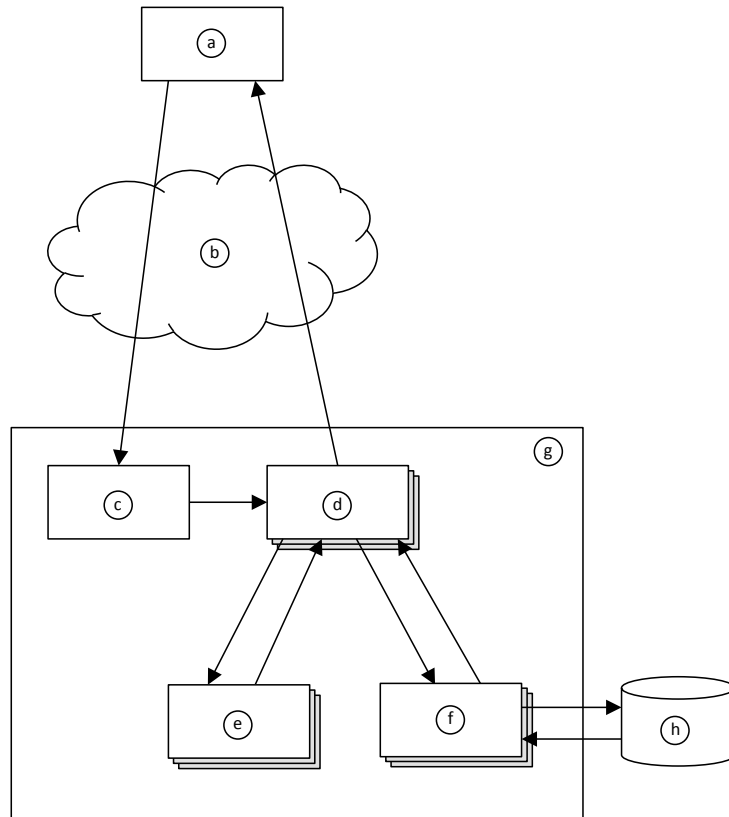
This command generates an appropriate migration file. Note that the class name after migration must be of the form `AddXxxToYyy`.

- b. The view (“V”) files above (i.e., the ERBs) would need to also display the songwriter values by adding appropriate HTML and calls to `song.songwriter`.
- c. In the controller (“C”) file above (`song_controller.rb`), the `song_params` method would need to be updated to account for the `:songwriter` parameter.

Problem:

Given the Rails MVC architectural diagram below, label each component.

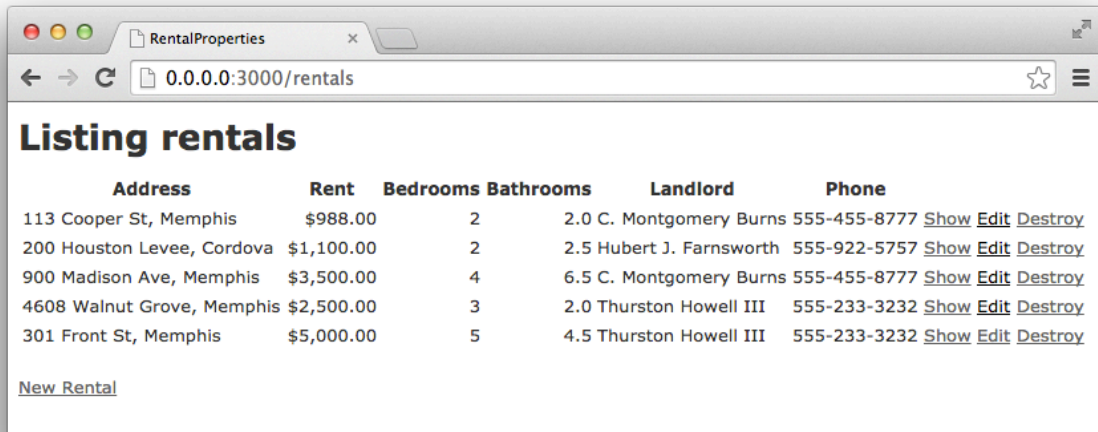
1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____



Solution:

1. Web Browser
2. Ye Olde Internet
3. Rails Router
4. Controller
5. View
6. Model
7. Rails Server
8. Database

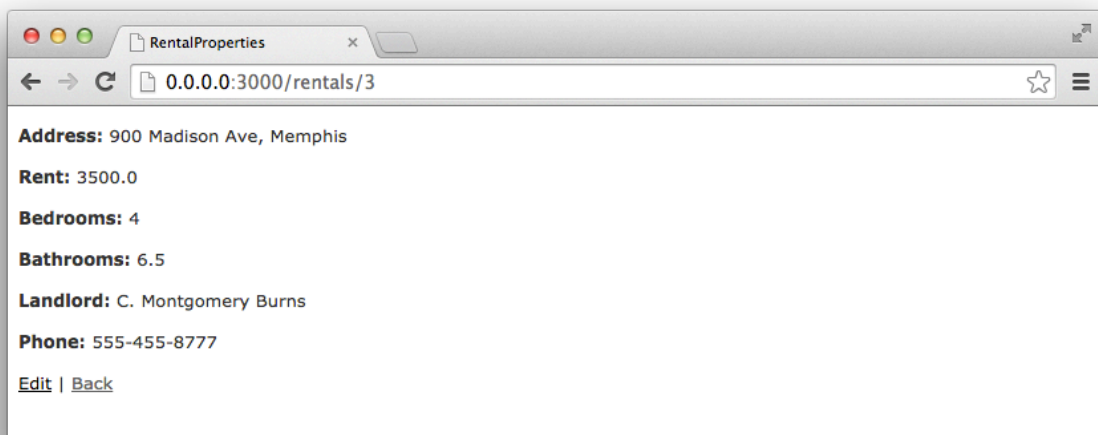
Here are some figures to consider while answering the following questions.



Address	Rent	Bedrooms	Bathrooms	Landlord	Phone
113 Cooper St, Memphis	\$988.00	2	2.0	C. Montgomery Burns	555-455-8777
200 Houston Levee, Cordova	\$1,100.00	2	2.5	Hubert J. Farnsworth	555-922-5757
900 Madison Ave, Memphis	\$3,500.00	4	6.5	C. Montgomery Burns	555-455-8777
4608 Walnut Grove, Memphis	\$2,500.00	3	2.0	Thurston Howell III	555-233-3232
301 Front St, Memphis	\$5,000.00	5	4.5	Thurston Howell III	555-233-3232

[New Rental](#)

Figure 8. Index page for rental-property web app.



Address: 900 Madison Ave, Memphis

Rent: 3500.0

Bedrooms: 4

Bathrooms: 6.5

Landlord: C. Montgomery Burns

Phone: 555-455-8777

[Edit](#) | [Back](#)

Figure 9. Show-rental page for rental-property web app.


```

$ rake routes
  Prefix Verb  URI Pattern          Controller#Action
  rentals GET   /rentals(.:format)  rentals#index
                   POST  /rentals(.:format)  rentals#create
  new_rental GET  /rentals/new(.:format) rentals#new
  edit_rental GET  /rentals/:id/edit(.:format) rentals#edit
  rental GET   /rentals/:id(.:format) rentals#show
                   PATCH /rentals/:id(.:format) rentals#update
                   PUT   /rentals/:id(.:format) rentals#update
                   DELETE /rentals/:id(.:format) rentals#destroy

```

Figure 10. Result of "rake routes" command for rental-property web app.

```

1  # == Schema Information
2  #
3  # Table name: rentals
4  #
5  # id          :integer          not null, primary key
6  # address     :string(255)
7  # rent        :decimal(, )
8  # bedrooms   :integer
9  # bathrooms   :float
10 # landlord    :string(255)
11 # phone       :string(255)
12 # created_at  :datetime
13 # updated_at  :datetime
14 #
15
16 class Rental < ActiveRecord::Base
17 end

```

Figure 11. Rental-property web app file: app/models/rental.rb

```

1 ▼ class RentalsController < ApplicationController
2   def index
3     @rentals = Rental.all
4   end
5
6   def show
7     # YOUR ANSWER HERE
8   end
9
10  def new
11    @rental = Rental.new
12  end
13
14  def edit
15    @rental = Rental.find(params[:id])
16  end
17
18 ▼ def create
19   @rental = Rental.new(rental_params)
20 ▼   respond_to do |format|
21 ▼     if @rental.save
22       format.html { redirect_to @rental, notice: 'Rental was successfully created.' }
23       format.json { render action: 'show', status: :created, location: @rental }
24 ▼     else
25       format.html { render action: 'new' }
26       format.json { render json: @rental.errors, status: :unprocessable_entity }
27     end
28   end
29 end
... and so on ...

```

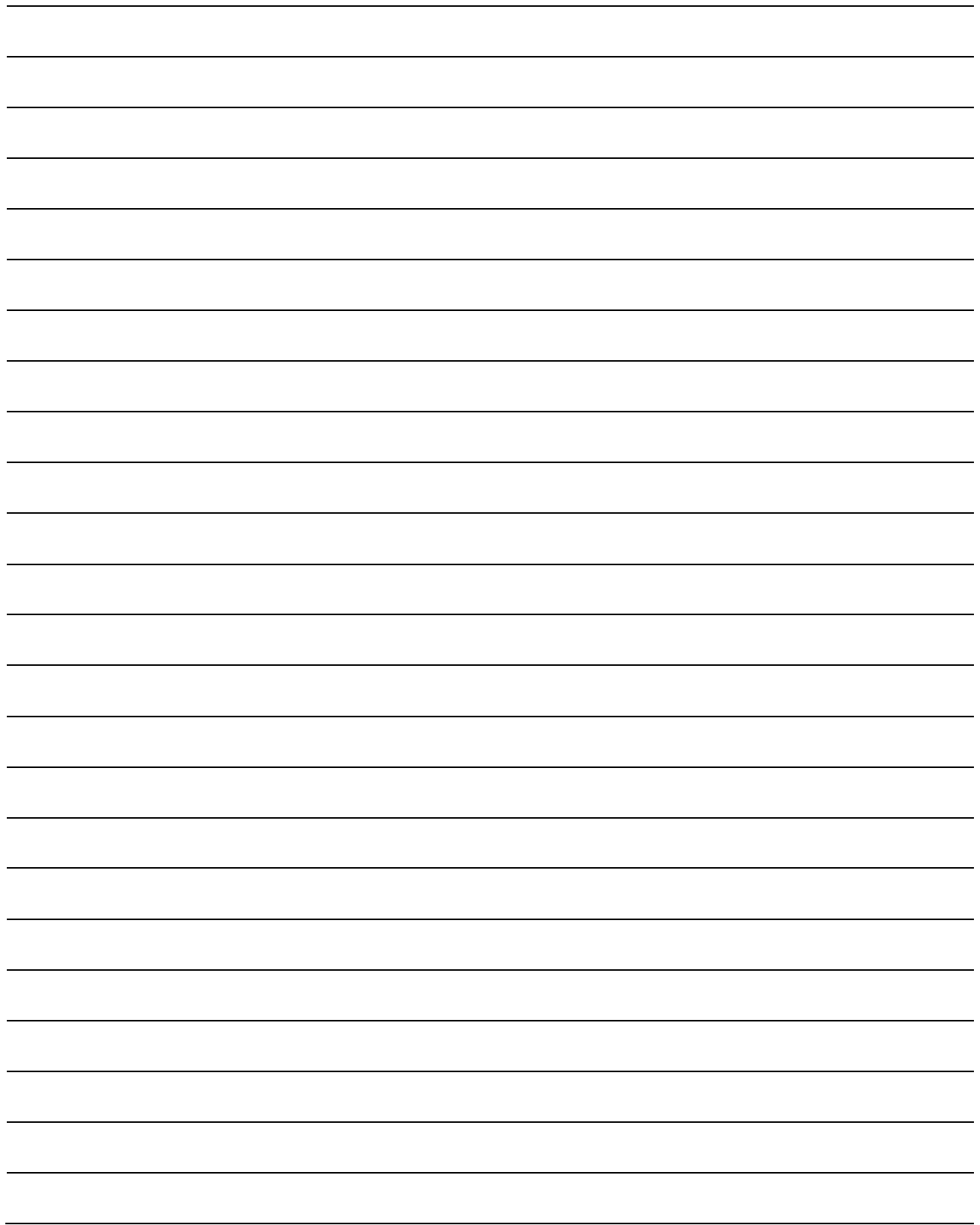
Figure 12. Rental-property web app file: app/controllers/rentals_controller.rb

```

1 <h1>Listing rentals</h1>
2
3 <table>
4   <thead>
5     <tr>
6       <th>Address</th>
7       <th>Rent</th>
8       <th>Bedrooms</th>
9       <th>Bathrooms</th>
10      <th>Landlord</th>
11      <th>Phone</th>
12      <th></th>
13      <th></th>
14      <th></th>
15    </tr>
16  </thead>
17
18  <tbody>
19    <%= @rentals.each do |rental| %>
20      <tr>
21        <td><%= rental.address %></td>
22        <td style="text-align: right;"><%= number_to_currency(rental.rent) %></td>
23        <td style="text-align: right;"><%= rental.bedrooms %></td>
24        <td style="text-align: right;"><%= rental.bathrooms %></td>
25        <td><%= rental.landlord %></td>
26        <td><%= rental.phone %></td>
27        <td><%= link_to 'Show', rental %></td>
28        <td><%= link_to 'Edit', edit_rental_path(rental) %></td>
29        <td><%= link_to 'Destroy', rental, method: :delete, data: { confirm: 'Are you sure?' } %>
30      </td>
31    </tr>
32    <%= end %>
33  </tbody>
34 </table>
35
36 <br>
37 <%= link_to 'New Rental', new_rental_path %>

```

Figure 13. Rental-property web app file: app/views/index.html.erb



Solution:

```
def show
  @rental = Rental.find(params[:id])
end
```

<p>

Address: <%= @rental.address %>

</p>

<p>

Rent: <%= @rental.rent %>

</p>

<p>

Bedrooms: <%= @rental.bedrooms %>

</p>

<p>

Bathrooms: <%= @rental.bathrooms %>

</p>

<p>

Landlord: <%= @rental.landlord %>

</p>

<p>

Phone: <%= @rental.phone %>

</p>

cont'd next page

<%= link_to 'Edit', edit_rental_path(@rental) %>

<%= link_to 'Back', rentals_path %>

Solution:

It would violate the single-responsibility principle (SRP) because a controller is responsible for translating between UI actions and operations on the model, whereas a view is responsible for UI presentation. Line 3 is an operation on the model — a controller responsibility. Moving this line into the view would mean that the view now has both view and controller responsibilities.

Here is a figure to consider while answering the following questions.

```
1 # id      :integer      not null, primary key
2 # name    :string
3 # email   :string
4 class User < ActiveRecord::Base
5   has_many :sales
6 end
```

```
1 # id      :integer      not null, primary key
2 class Sale < ActiveRecord::Base
3   belongs_to :user
4   has_many :line_items
5 end
```

```
1 # id      :integer      not null, primary key
2 # quantity :integer
3 class LineItem < ActiveRecord::Base
4   belongs_to :sale
5   belongs_to :item_description
6 end
```

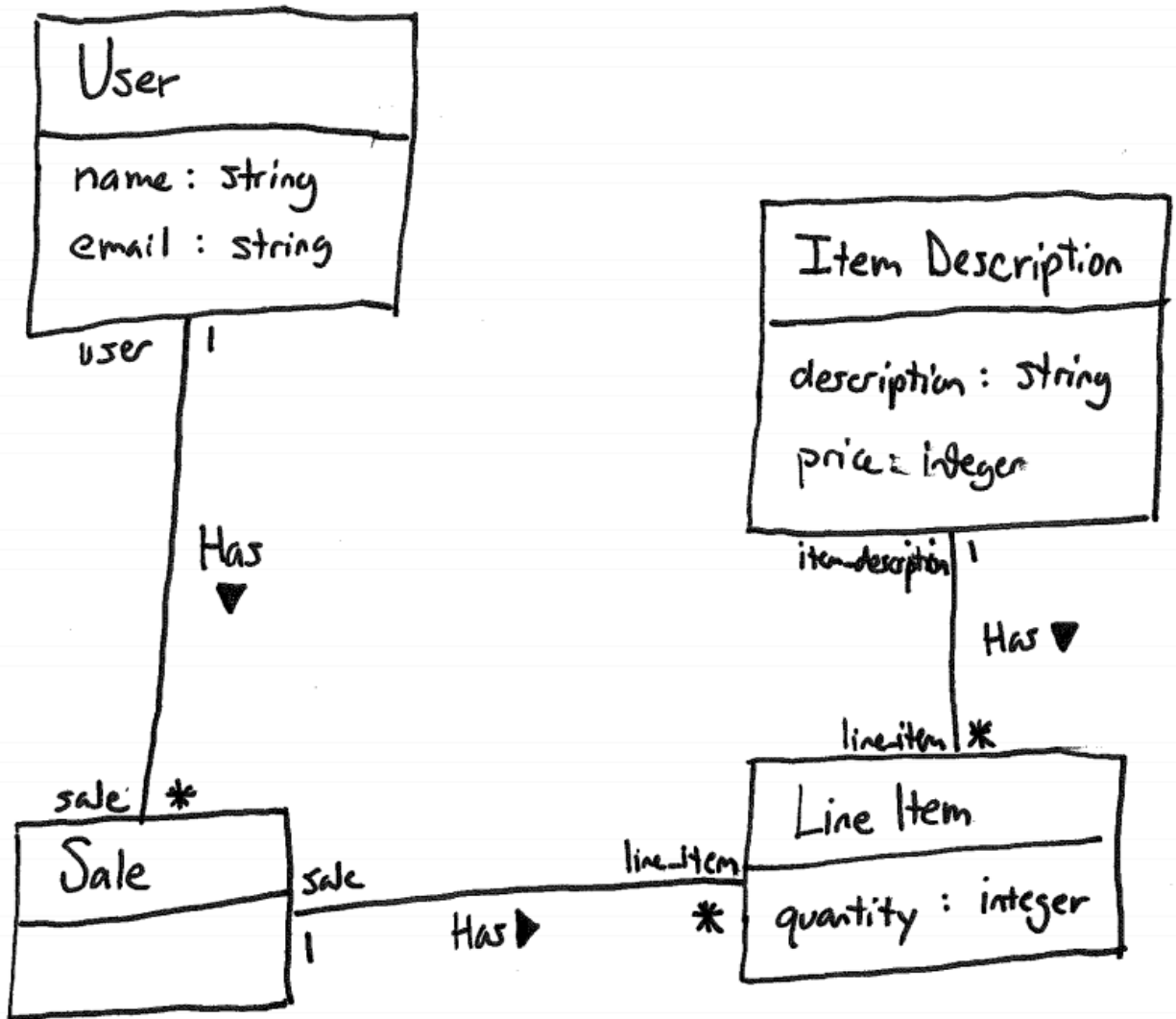
```
1 # id      :integer      not null, primary key
2 # description :string
3 # price    :integer
4 class ItemDescription < ActiveRecord::Base
5   has_many :line_items
6 end
```

Figure 14. Model classes for a point-of-sale system.

Problem:

Create a UML class diagram representing the Figure 14 point-of-sale model classes. Be sure to label all associations and association ends, and include all multiplicities. Don't include "id" attributes (objects have identity by default).

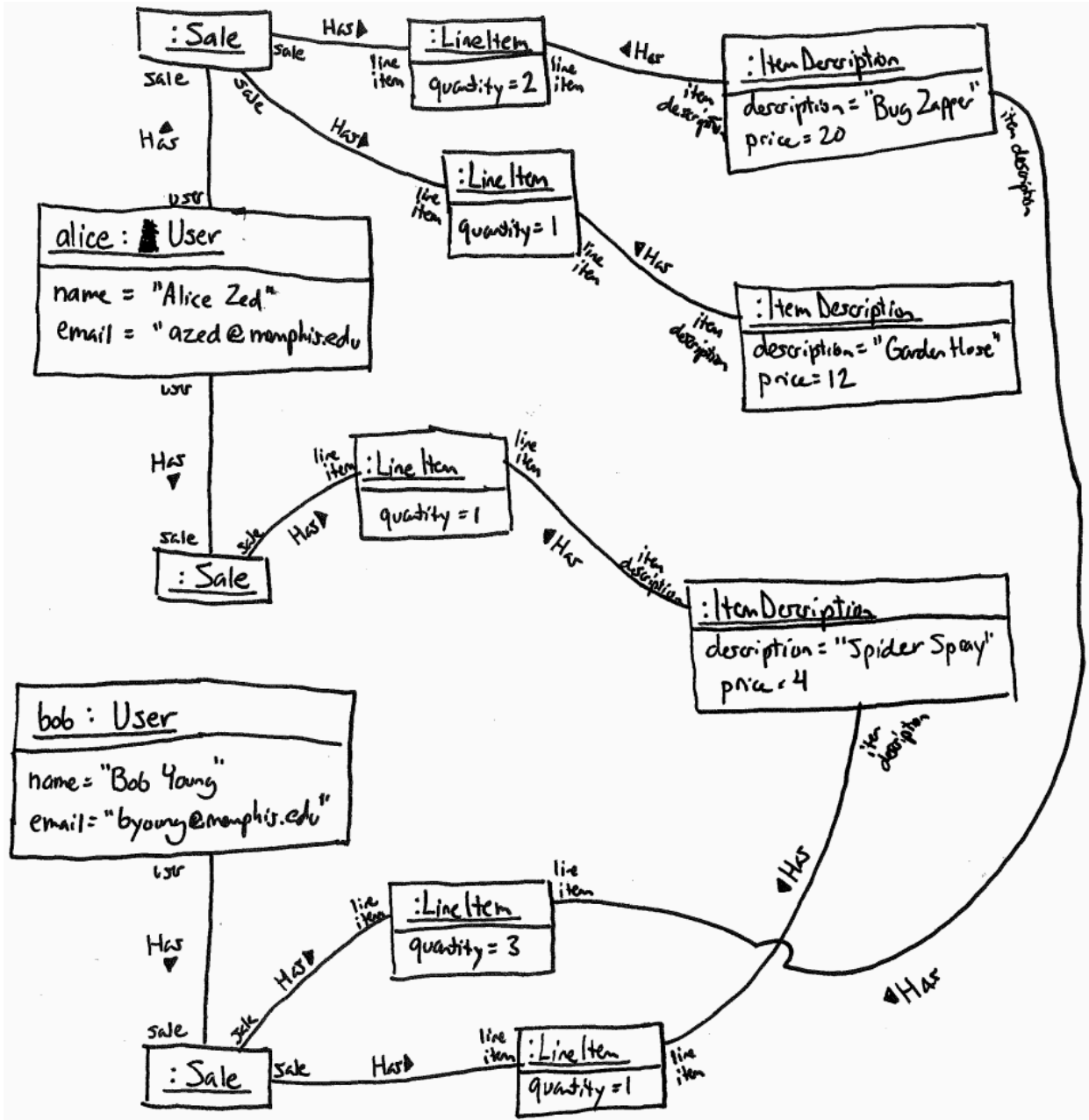
Solution:



Problem:

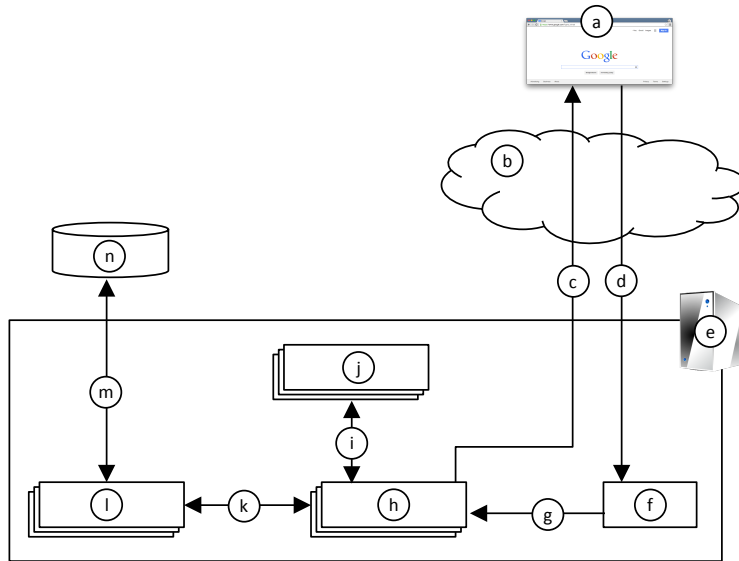
Consider the following execution of a point-of-sale system with the model in Figure 14. Two users register: Alice Zed (azed@memphis.edu) and Bob Young (byoung@memphis.edu). Alice purchases the following things: 2 Bug Zappers (\$20 each) and 1 Garden Hose (\$12 each). Bob purchases the following things: 3 Bug Zappers and 1 Spider Spray (\$4 each). Later, Alice makes another purchase: 1 Spider Spray. Create an object diagram that depicts the model objects after this execution.

Solution:



Problem:

Consider this architectural diagram:



For each lettered item, fill in the most appropriate label number.

- | | |
|----------|-------------------------------------|
| a. _____ | 1) Ye Olde Internet |
| b. _____ | 2) Invocation of Model Operations |
| c. _____ | 3) Rails Controller |
| d. _____ | 4) Rendering of View |
| e. _____ | 5) SQL Queries |
| f. _____ | 6) Relational Database |
| g. _____ | 7) HTTP Response |
| h. _____ | 8) Rails Server |
| i. _____ | 9) Web Browser |
| j. _____ | 10) Rails Router |
| k. _____ | 11) Invocation of Controller Action |
| l. _____ | 12) Rails View |
| m. _____ | 13) HTTP Request |
| n. _____ | 14) Rails Model |

Solutions:

- a. 9
- b. 1
- c. 7
- d. 13
- e. 8
- f. 10
- g. 11
- h. 3
- i. 4
- j. 12
- k. 2
- l. 14
- m. 5
- n. 6

- 1) Ye Olde Internet
- 2) Invocation of Model Operations
- 3) Rails Controller
- 4) Rendering of View
- 5) SQL Queries
- 6) Relational Database
- 7) HTTP Response
- 8) Rails Server
- 9) Web Browser
- 10) Rails Router
- 11) Invocation of Controller Action
- 12) Rails View
- 13) HTTP Request
- 14) Rails Model

The questions on the following pages refer to the example figures below. The figures show different aspects of a WeddingHelper web app that helps a wedding planner keep track of which guests have been sent invitations and thank-you letters, and what gifts the couple received from each guest. Because each correspondence (e.g., invitation) is often sent to a household of multiple people (such as a married couple) and each gift typically comes from all the people in a household, the system organizes the guests as a set of households, each made up of one or more people.

The system has three model classes, Household, Person, and Gift (see Figure 15) and a controller class for each (not shown). Figure 16 and Figure 17 show what the index pages for households and gifts, respectively, look like. Figure 18 and Figure 19 show the ERB code for each index page (partially elided in the case of Figure 19). Figure 20 shows partially elided test code for the Person model class, and Figure 21 a form for creating a new person. (Note that Rails knows that the plural of *person* is *people*.)

```

# Table name: households
#
# id          :integer          not null, primary key
# invitation_sent :boolean
# thankyou_sent :boolean
# created_at   :datetime        not null
# updated_at   :datetime        not null
#
class Household < ActiveRecord::Base
  has_many :people
  has_many :gifts
end

```

```

# Table name: people
#
# id          :integer          not null, primary key
# name        :string
# email       :string
# created_at  :datetime        not null
# updated_at  :datetime        not null
# household_id :integer
#
class Person < ActiveRecord::Base
  belongs_to :household
  validates :name, presence: true
end

```

```

# Table name: gifts
#
# id          :integer          not null, primary key
# name        :string
# description  :text
# has_receipt :boolean
# estimated_value :integer
# created_at  :datetime        not null
# updated_at  :datetime        not null
# household_id :integer
#
class Gift < ActiveRecord::Base
  belongs_to :household
end

```

Figure 15. Model classes for Wedding Helper web app.

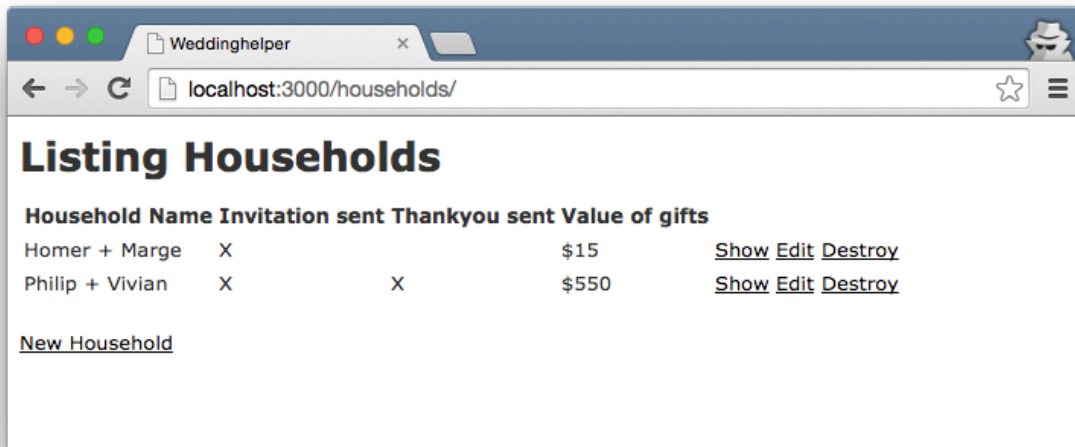


Figure 16. Index page for households.

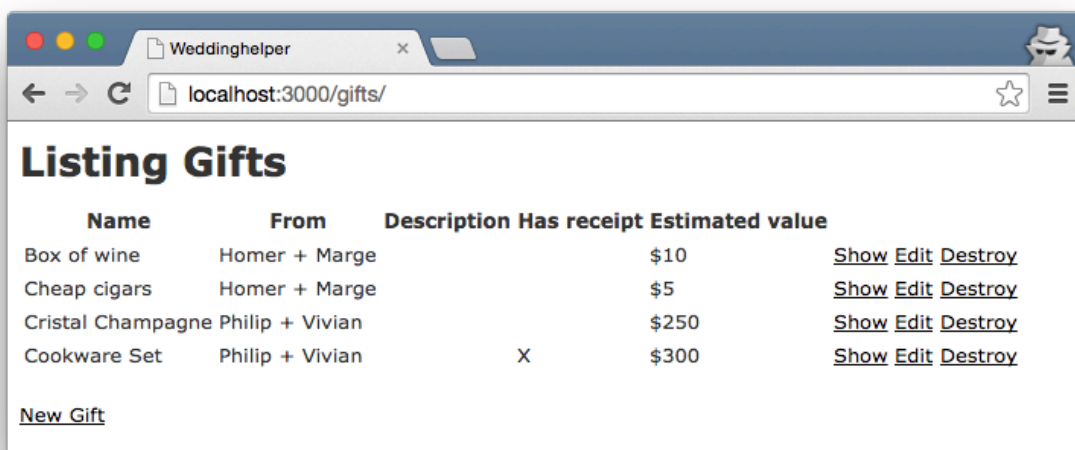


Figure 17. Index page for gifts. Note that the "Description" attribute happens to have been left empty in all cases.

```

<p id="notice"><%= notice %></p>

<h1>Listing Households</h1>

<table>
  <thead>
    <tr>
      <th>Household Name</th>
      <th>Invitation sent</th>
      <th>Thankyou sent</th>
      <th>Value of gifts</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @households.each do |household| %>
      <tr>
        <td>
          <% household.people.each do |person| %>
            <%= person.name %>
            <% if person != household.people.last %>
              +
            <% end %>
          <% end %>
        </td>
        <td><% if household.invitation_sent %>X<% end %></td>
        <td><% if household.thankyou_sent %>X<% end %></td>
        <td>
          <%
            gift_total = 0
            household.gifts.each do |gift|
              gift_total += gift.estimated_value
            end
          %>
          $<%= gift_total %>
        </td>
        <td><%= link_to 'Show', household %></td>
        <td><%= link_to 'Edit', edit_household_path(household) %></td>
        <td><%= link_to 'Destroy', household, method: :delete, data: { confirm:
          'Are you sure?' } %></td>
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<%= link_to 'New Household', new_household_path %>

```

Figure 18. View code for households index page.

```
<p id="notice"><%= notice %></p>

<h1>Listing Gifts</h1>

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>From</th>
      <th>Description</th>
      <th>Has receipt</th>
      <th>Estimated value</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <div style="border: 1px solid black; background-color: #cccccc; padding: 20px; text-align: center; width: fit-content; margin: 10px auto; min-height: 200px;">
      Fill in this code
    </div>
  </tbody>
</table>

<br>

<%= link_to 'New Gift', new_gift_path %>
```

Figure 19. Partially elided view code for gifts index page.

```
require 'test_helper'

class PersonTest < ActiveSupport::TestCase

  def setup
    @person = Person.new(name: "Homer", email: "homer@example.com")
  end

  test "name should be present" do
    

Fill in this code


  end

end
```

Figure 20. Model test case with elided code.

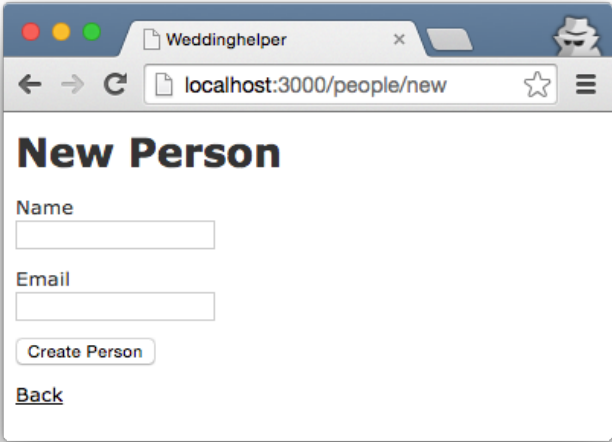
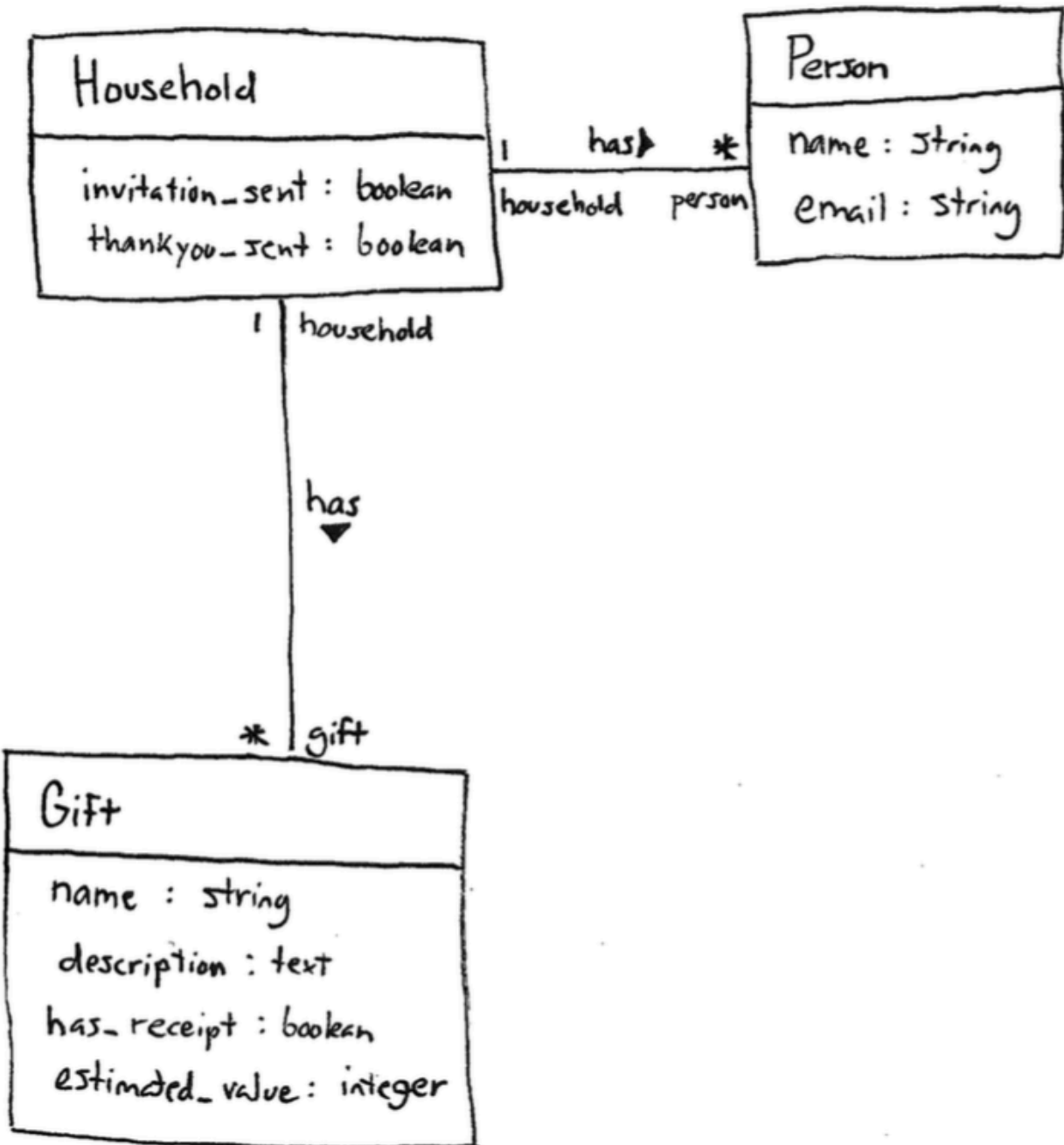


Figure 21. Form for creating a new person.

Problem:

Draw a UML class diagram that represents the model classes given in Figure 15. Be sure to label all associations and association ends, and include all multiplicities. Don't include "id" attributes (objects have identity by default). You may also omit the datetime attributes.

Solution:



Problem:

Write the missing ERB code in Figure 19 such that it renders pages that look like Figure 17. Do not hard code values. Rather, they should come from an `@gifts` object that was passed to the ERB. In particular, `@gifts` is an array of Gift objects.

Solution:

```
<% @gifts.each do |gift| %>
  <tr>
    <td><%= gift.name %></td>
    <td>
      <% gift.household.people.each do |person| %>
        <%= person.name %>
        <% if person != gift.household.people.last %>
          +
        <% end %>
      <% end %>
    </td>
    <td><%= gift.description %></td>
    <td><% if gift.has_receipt %>X<% end %></td>
    <td><%= gift.estimated_value %></td>
    <td><%= link_to 'Show', gift %></td>
    <td><%= link_to 'Edit', edit_gift_path(gift) %></td>
    <td><%= link_to 'Destroy', gift, method: :delete, data: { confirm: 'Are you sure?' } %>
      </td>
  </tr>
<% end %>
```

Problems:

1. In the household index view, `@households` is an array of all the household objects. In what method was that array populated? Give the class name and method name. (These aren't shown anywhere in this exam, but you should be able to make a sensible guess.)

2. Fill in the missing test code in Figure 20 such that the test checks that the model class' validation features will catch a missing name. Recall that all Rails model classes have a `valid?` method, and the test base class provides `assert` and `assert_not` methods.

Solutions:

1.

HouseholdsController # index

2.

```
@person.name = ""  
assert_not @person.valid?
```

Multiple-Choice Questions:

1. If you wanted to change the HTTP request URL that maps to a particular controller action, which Rails component would you need to modify?
 - a. Controller class
 - b. Model class
 - c. Routes class
 - d. Migration class
 - e. All of the above

2. Which of the following types of Rails components sets up the database tables?
 - a. Controller classes
 - b. Model classes
 - c. Routes classes
 - d. Migration classes
 - e. All of the above

3. What type of HTTP request would be generated by pressing the “Create Person” button in the form in Figure 21.
 - a. GET
 - b. POST
 - c. PATCH
 - d. DELETE
 - e. None of the above

4. After the HTTP request generated by Figure 21 is successfully processed on the server side, what should the server’s response to the browser be?
 - a. HTTP response with successful status and accompanying HTML
 - b. HTTP response with unsuccessful status (404 Not Found) and no HTML
 - c. HTTP redirect to another URL
 - d. No response
 - e. None of the above

Solutions:

1. c

2. d

3. b

4. c

The questions on the following pages refer to these example figures. The figures show different aspects of the *MeetMe* web app that enables people to post “meetup” opportunities to “boards”. Each city has its own board with one person who serves as coordinator.

```
# == Schema Information
#
# Table name: boards
#
# id          :integer          not null, primary key
# city       :string
# coordinator_name :string
# coordinator_email :string
# created_at  :datetime         not null
# updated_at  :datetime         not null
#
class Board < ActiveRecord::Base
  has_many :meetups
  validates :city, presence: true
  validates :coordinator_name, presence: true
  validates :coordinator_email, presence: true
end
```

```
# == Schema Information
#
# Table name: meetups
#
# id          :integer          not null, primary key
# who        :string
# where      :string
# when       :datetime
# created_at :datetime         not null
# updated_at :datetime         not null
# board_id   :integer
#
class Meetup < ActiveRecord::Base
  belongs_to :board
  validates :who, presence: true
  validates :where, length: { minimum: 3 }
  validates :when, presence: true
end
```

Figure 22. Model classes for the MeetMe web app.

```
mcdonalds:  
  who: Ronald McDonald  
  where: McDonald's  
  when: 2015-10-10 22:00:00
```

```
subway:  
  who: Jared Fogle  
  where: Subway  
  when: 2015-10-26 12:30:00
```

```
require 'test_helper'
```

```
class MeetupTest < ActiveSupport::TestCase
```

```
  # test "the truth" do  
  #   assert true  
  # end
```

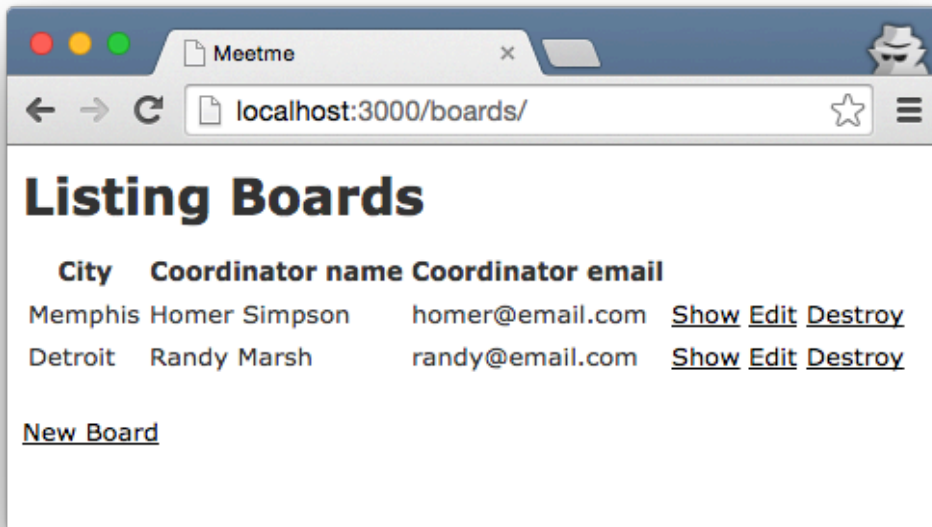
```
  test "where should be longer than 3 characters" do
```

Fill in this code

```
end
```

```
end
```

Figure 23. Test fixture (upper) and test case (lower). [Oops. The test string should say “at least 3 characters”.]

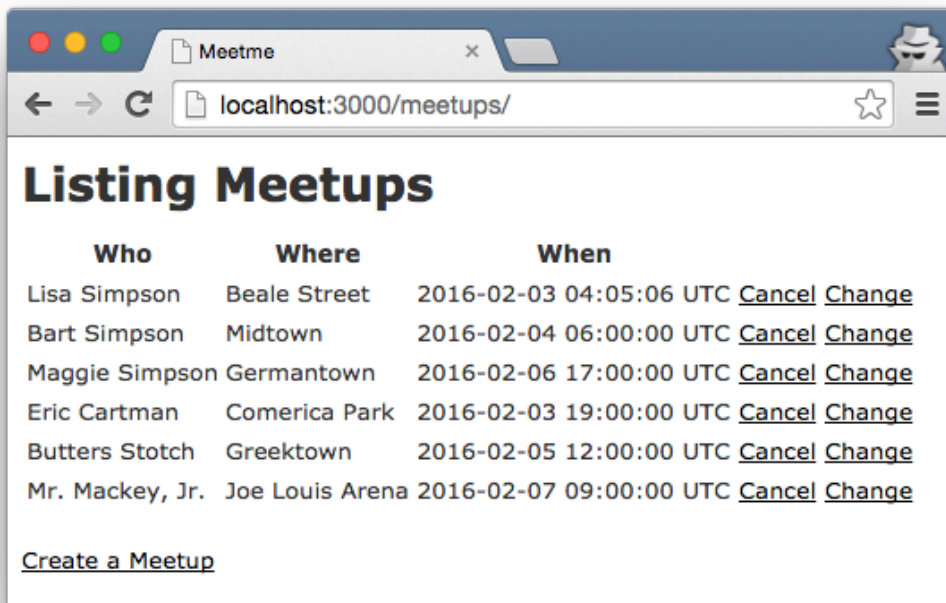


```

<h1>Listing Boards</h1>
<table>
  <thead>
    <tr>
      <th>City</th>
      <th>Coordinator name</th>
      <th>Coordinator email</th>
      <th colspan="3"></th>
    </tr>
  </thead>
  <tbody>
    <% @boards.each do |board| %>
      <tr>
        <td><%= board.city %></td>
        <td><%= board.coordinator_name %></td>
        <td><%= board.coordinator_email %></td>
        <td><%= link_to 'Show', board %></td>
        <td><%= link_to 'Edit', edit_board_path(board) %></td>
        <td><%= link_to 'Destroy', board, method: :delete, data: { confirm: 'Are you sure?' } %></td>
      </tr>
    <% end %>
  </tbody>
</table>
<br>
<%= link_to 'New Board', new_board_path %>

```

Figure 24. "index" page for the Board model class.



```

<p id="notice"><%= notice %></p>

<h1>Listing Meetups</h1>

<table>
  <thead>
    <tr>
      <th>Who</th>
      <th>Where</th>
      <th>When</th>
      <th colspan="2"></th>
    </tr>
  </thead>

  <tbody>
    <tr>
      <td><%= @meetup.name %></td>
      <td><%= @meetup.location %></td>
      <td><%= @meetup.start_time %></td>
      <td><%= @meetup.cancel_path %></td>
      <td><%= @meetup.change_path %></td>
    </tr>
  </tbody>
</table>

<br>

<%= link_to 'Create a Meetup', new_meetup_path %>

```

Figure 25. "index" view for the Meetup model class. "Cancel" deletes a meetup, and "Change" links to an edit form.

Meetme

localhost:3000/meetups/new

New Meetup

Who

Where

When
2015 October 7 04 : 18

[Back](#)

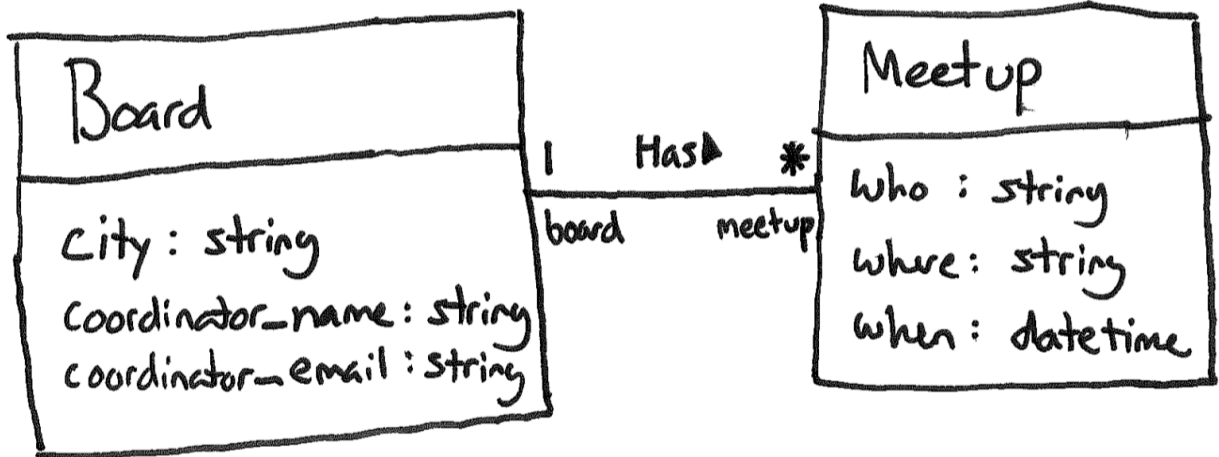
Figure 26. The form for creating a new meetup.

Problem:

Draw a UML class diagram that represents the model classes given in Figure 22. Be sure to label all associations and association ends, and include all multiplicities. Don't include "id" attributes (objects have identity by default). You may also omit the "datetime" attributes that Rails provides by default.

Solution:

Draw a UML class diagram that represents the model classes given in Figure 22. Be sure to label all associations and association ends, and include all multiplicities. Don't include "id" attributes (objects have identity by default). You may also omit the "datetime" attributes that Rails provides by default.



Problem:

Fill in the missing test code in Figure 23 such that the test checks that the model class' validation will catch a "where" attribute that has too few characters. Recall that all Rails model classes have a `valid?` method, and the test base class provides `assert` and `assert_not` methods. Also, you can retrieve a model fixture object with a line like this:

```
subway = meetups(:subway)
```

Solution:

Fill in the missing test code in Figure 23 such that the test checks that the model class' validation will catch a "where" attribute that has too few characters. Recall that all Rails model classes have a `valid?` method, and the test base class provides `assert` and `assert_not` methods. Also, you can retrieve a model fixture object with a line like this:

```
subway = meetups(:subway)
```

```
subway = meetups(:subway)
subway.where = "X"
assert_not subway.valid?
```

Problem:

Write the missing ERB code in Figure 25 such that it renders pages that look like the page depicted in the figure. Do not hard code values. Rather, they should come from an `@meetups` object that is passed to the ERB. In particular, `@meetups` is an array of `Meetup` objects.

Solution:

Write the missing ERB code in Figure 25 such that it renders pages that look like the page depicted in the figure. Do not hard code values. Rather, they should come from an `@meetups` object that is passed to the ERB. In particular, `@meetups` is an array of `Meetup` objects.

```
<% @meetups.each do |meetup| %>
  <tr>
    <td><%= meetup.who %></td>
    <td><%= meetup.where %></td>
    <td><%= meetup.when %></td>
    <td><%= link_to 'Cancel', meetup, method: :delete %></td>
    <td><%= link_to 'Change', edit_meetup_path(meetup) %></td>
  </tr>
<% end %>
```

Multiple-Choice Questions:

1. What type of HTTP request would be generated by pressing the “Create Meetup” button on the form in Figure 26.
 - a. GET
 - b. POST
 - c. PATCH
 - d. DELETE
 - e. None of the above
2. Which of the following lines of code would the `MeetupsController#index` action contain?
 - a. `@meetup = Meetup.new`
 - b. `@meetup = Meetup.find(params[:id])`
 - c. `@meetup = Meetup.new(meetup_params)`
 - d. `@meetups = Meetup.all`
 - e. None of the above
3. Which of the following lines of code would the `MeetupsController#new` action likely contain?
 - a. `@meetup = Meetup.new`
 - b. `@meetup = Meetup.find(params[:id])`
 - c. `@meetup = Meetup.new(meetup_params)`
 - d. `@meetups = Meetup.all`
 - e. None of the above

4. True or false? Controller actions that modify the database (such as the `create` action) should end by sending an HTTP redirect response to the browser (instead of rendering an HTML page to send in the response).
 - a. True
 - b. False

Solutions:

1. b

2. d

3. a

4. a

The questions on the following pages refer to the example figures. The figures show different aspects of the *Warrior World* web app that is a roleplaying adventure game thematically similar to *Dungeons & Dragons* and *World of Warcraft*. In the game, users play as heroes, each with his/her own back story (e.g., land of origin) and special weapons and equipment.

```
# == Schema Information
#
# Table name: heros
#
# id          :integer          not null, primary key
# name       :string
# race       :string
# hit_points :integer
# created_at :datetime         not null
# updated_at :datetime         not null
# home_land_id :integer
#

class Hero < ActiveRecord::Base
  has_many :equipment
  belongs_to :home_land

  validates :name, presence: true
  validates :race, presence: true
  validates :hit_points, numericality: { greater_than_or_equal_to: 0}
end

# == Schema Information
#
# Table name: equipment
#
# id          :integer          not null, primary key
# name       :string
# description :string
# created_at :datetime         not null
# updated_at :datetime         not null
# hero_id    :integer
#

class Equipment < ActiveRecord::Base
  belongs_to :hero

  validates :name, presence: true
  validates :description, presence: true
end

# == Schema Information
#
# Table name: home_lands
#
# id          :integer          not null, primary key
# name       :string
# geography  :string
# created_at :datetime         not null
# updated_at :datetime         not null
#

class HomeLand < ActiveRecord::Base
  has_many :hero

  validates :name, presence: true
  validates :geography, presence: true
end
```

Figure 27. Three model classes from Warrior World.

```
alice:  
  name: Alice the Fire Angel  
  race: Human  
  hit_points: 88  
  
archimonde:  
  name: Archimonde the Defiler  
  race: Orcs  
  hit_points: 108
```

Figure 28. Test fixture for class Hero.

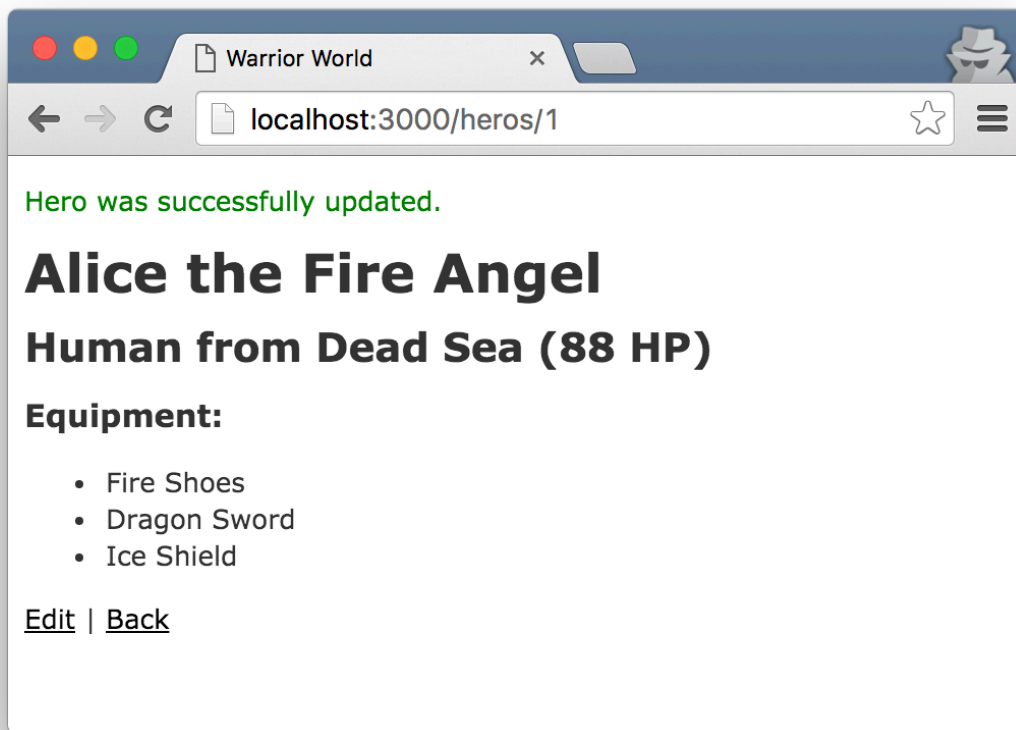
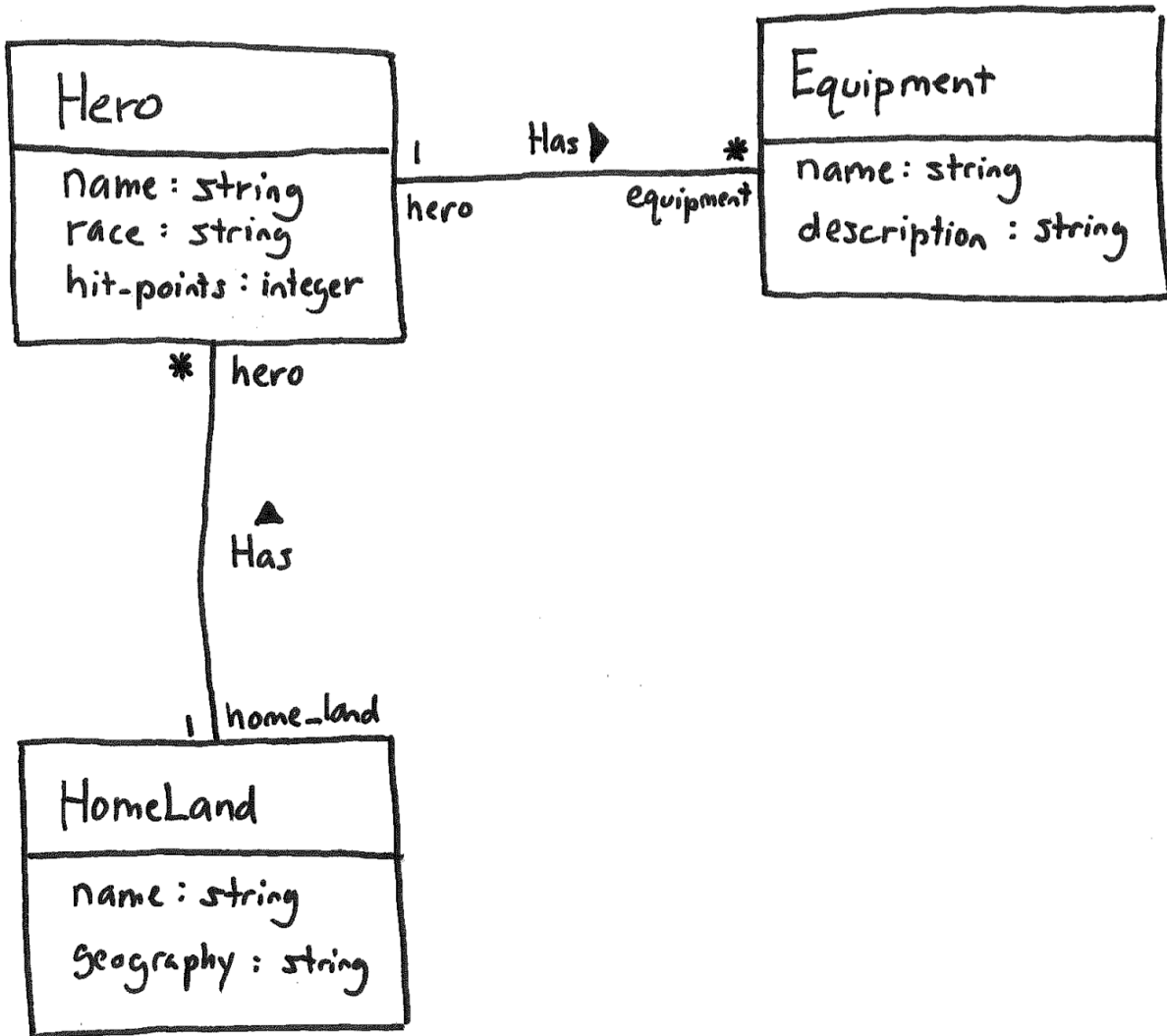


Figure 29. Hero *show* page.

Problem:

Draw a UML class diagram that represents the three model classes given in Figure 27. Be sure to label all associations and association ends, and include all multiplicities. Don't include any "id" attributes (including foreign keys). You may also omit the "datetime" attributes that Rails provides by default.

Solution:



Problem:

Consider the validations in the Hero class (Figure 27) and the Hero fixtures in Figure 28. Using the following lines of code, create a class with two test cases—one that tests that name is present and the other that tests that hit_points are 0 or greater. You should use all lines at least once, and some lines may be used more than once.

- a) `archimonde = heros(:archimonde)`
- b) `test "hit points should be greater than or equal to 0" do`
- c) `test "name should not be empty" do`
- d) `class HeroTest < ActiveSupport::TestCase`
- e) `alice = heros(:alice)`
- f) `assert alice.invalid?`
- g) `end`
- h) `assert archimonde.invalid?`
- i) `archimonde.name = nil`
- j) `alice.hit_points = -1`

1) _____

2) _____

3) _____

4) _____

5) _____

6) _____

7) _____

8) _____

9) _____

10) _____

11) _____

12) _____

Solutions:

1) d

2) b

3) e

4) j

5) f

6) g

7) c

8) a

9) i

10) h

11) g

12) g



Problem:

Consider the Hero *show* page in Figure 29. Using the following lines of code, reverse engineer the view code that produced this page. You should use all lines at least once, and some lines may be used more than once.

- a. `<%= @hero.race %>`
- b. `</h2>`
- c. ` <%= equipment.name %> `
- d. ``
- e. ``
- f. `<h1><%= @hero.name %></h1>`
- g. `<p id="notice"><%= notice %></p>`
- h. `<% @hero.equipment.each do |equipment| %>`
- i. `from`
- j. `<%= link_to 'Back', heros_path %>`
- k. `(<%= @hero.hit_points %> HP)`
- l. `<% end %>`
- m. `<%= @hero.home_land.name %>`
- n. **Equipment:**
- o. `<h2>`
- p. `</h3>`
- q. `<h3>`
- r. `<%= link_to 'Edit', edit_hero_path(@hero) %> |`

1) _____

10) _____

2) _____

11) _____

3) _____

12) _____

4) _____

13) _____

5) _____

14) _____

6) _____

15) _____

7) _____

16) _____

8) _____

17) _____

9) _____

18) _____

Solutions:

1) g

2) F

3) o

4) a

5) i

6) m

7) K

8) b

9) q

10) n

11) p

12) d

13) h

14) c

15) l

16) e

17) r

18) j

Multiple-Choice Questions:

1. Which of the following routes corresponds to the *show* page in Figure 29?
 - a) `get '/heros', to: 'heros#index', as: 'heros'`
 - b) `get '/heros/:id/edit', to: 'heros#edit', as: 'edit_hero'`
 - c) `get '/heros/:id', to: 'heros#show', as: 'hero'`
 - d) `patch '/heros/:id', to: 'heros#update'`
 - e) `post '/heros', to: 'heros#create'`

2. Which of the following lines of code would the controller need to execute before rendering the Hero *show* view?
 - a) `@heros = Hero.all`
 - b) `@hero = Hero.new`
 - c) `@hero = Hero.new(params.require(:hero).permit(:name, :race, :hit_points))`
 - d) `@hero = Hero.find(params[:id])`
 - e) None of the above

3. True or false? State-affecting controller actions (such as create, update, and destroy) should always send an HTTP redirect response instead of rendering a view.
 - a) True
 - b) False

Solutions:

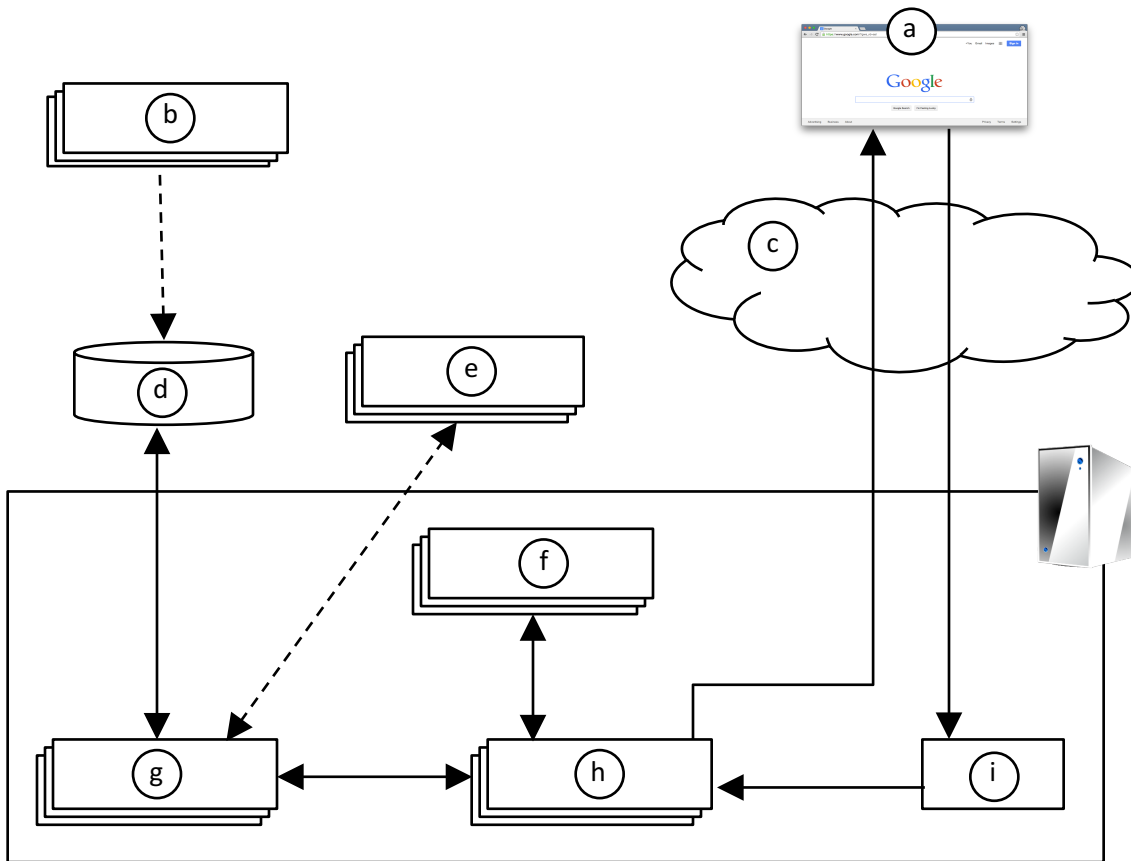
1. c

2. d

3. a

Problem:

Consider this architectural diagram in answering the following questions.



1. Which letter in the diagram corresponds to the code in Figure 27?
2. Which letter in the diagram corresponds to test cases?
3. Which letter in the diagram corresponds to view code?
4. Which letter in the diagram corresponds to routes code?
5. Which letter in the diagram corresponds to controller code?

Solutions:

1. g

2. e

3. f

4. i

5. h (would also accept g, h)

The questions on the following pages refer to the example figures. The figures show different aspects of the *find-a-dentist* web app that helps a patient to find a suitable dentist. Users can use the app to browse dentists and dental clinics, and to manage dentist and clinic data.

```
# == Schema Information
#
# Table name: clinics
#
# id              :integer          not null, primary key
# location        :string
# number_of_doctors :integer
# created_at      :datetime         not null
# updated_at      :datetime         not null
#
class Clinic < ApplicationRecord
  has_many :dentists
end

# == Schema Information
#
# Table name: dentists
#
# id              :integer          not null, primary key
# first_name     :string
# last_name      :string
# year_born      :integer
# created_at     :datetime         not null
# updated_at     :datetime         not null
# clinic_id      :integer
#
class Dentist < ApplicationRecord
  has_one :dentist_profile
  belongs_to :clinic
  validates :last_name, presence: true
  validates :year_born,
            numericality: { less_than_or_equal_to:
(Date.today.year - 17) }
end

# == Schema Information
#
# Table name: dentist_profiles
#
# id              :integer          not null, primary key
# birthplace     :string
# major          :string
# graduationyear :integer
# created_at     :datetime         not null
# updated_at     :datetime         not null
# dentist_id     :integer
#
class DentistProfile < ApplicationRecord
  belongs_to :dentist
end
```

Figure 30. Three model classes from the find-a-dentist app.

```
one:
  first_name: John
  last_name: Demento
  year_born: 1973

two:
  first_name: Sterling
  last_name: Bloodgush
  year_born: 1969
```

Figure 31. Test fixture for class Dentist.

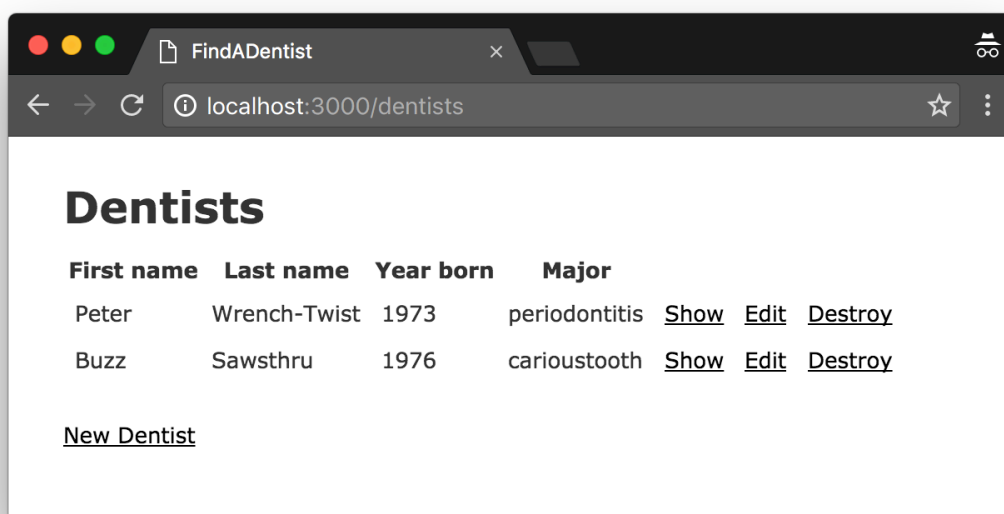
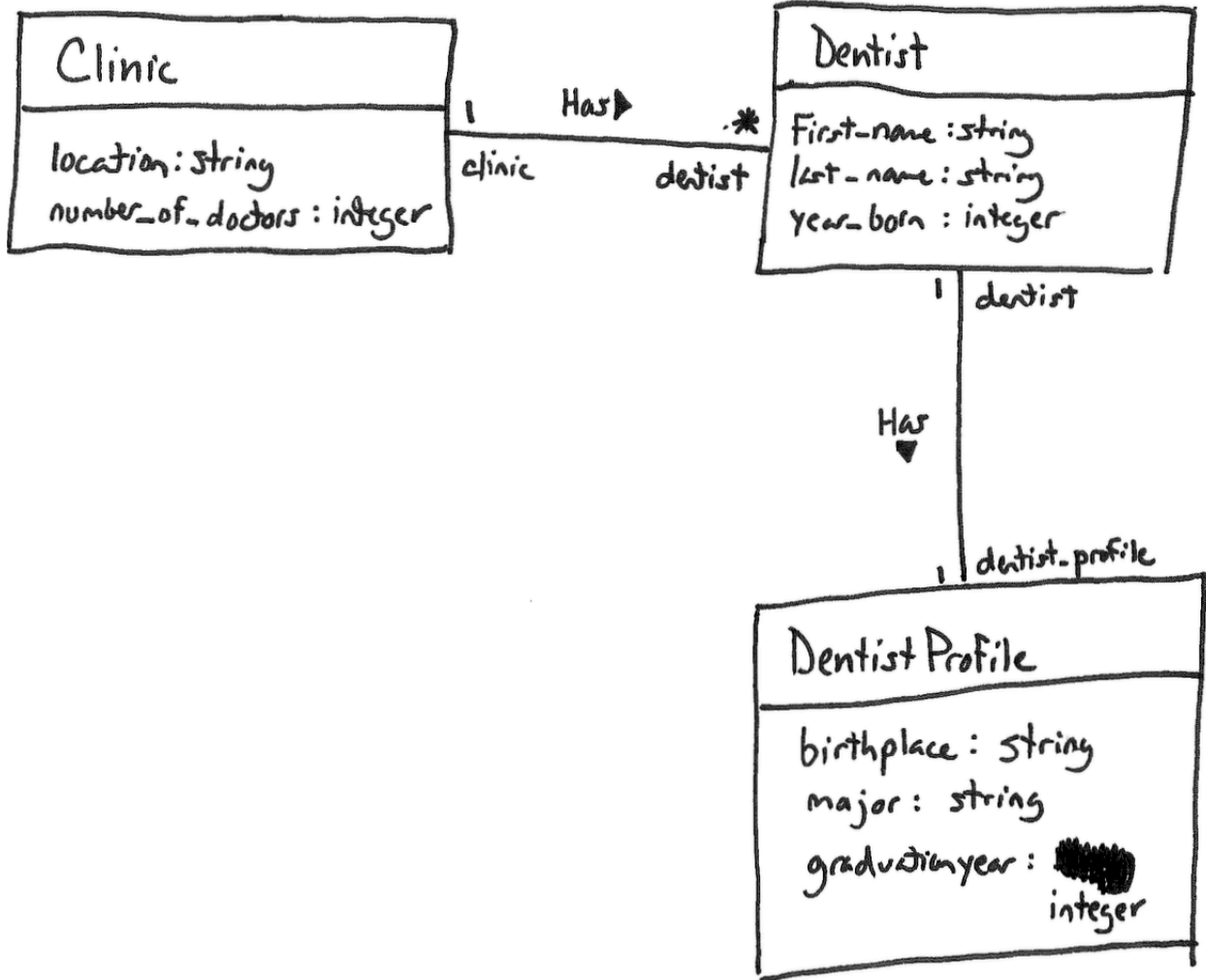


Figure 32. Dentist index page.

Problem:

Draw a UML class diagram that represents the three model classes given in Figure 30. Be sure to label all associations and association ends, and include all multiplicities. Don't include any "id" attributes (including foreign keys). You may also omit the "datetime" attributes that Rails provides by default.

Solution:



Problem:

Consider the validations in the Dentist class (Figure 30) and the Hero fixtures in Figure 31. Using the following lines of code, create a class with two test cases—one that tests that first name is present and the other that tests that the dentist is at least 17 or 18 years in age. You should use all lines at least once, and some lines may be used more than once.

- a) `assert_not one.valid?`
- b) `assert_not two.valid?`
- c) `class DentistTest < ActiveSupport::TestCase`
- d) `end`
- e) `one = dentists(:one)`
- f) `one.last_name = nil`
- g) `test "should be at least 17 or 18 years old" do`
- h) `test "should have a last name" do`
- i) `two = dentists(:two)`
- j) `two.year_born = Date.today.year`

1) _____

2) _____

3) _____

4) _____

5) _____

6) _____

7) _____

8) _____

9) _____

10) _____

11) _____

12) _____

Solution:

1) c

2) h

3) e

4) F

5) a

6) d

7) g

8) i

9) j

10) b

11) d

12) d



Either order

Problem:

Consider the Dentist *index* page in Figure 32. Using the following lines of code, reverse engineer the view code that produced this page. You should use all lines at least once, and some lines may be used more than once.

- a) <% @dentists.each do |dentist| %>
- b) <% end %>
- c) <%= link_to 'New Dentist', new_dentist_path %>
- d) </table>
- e) </tbody>
- f) </thead>
- g) </tr>
- h) <h1>Dentists</h1>
- i) <table>
- j) <tbody>
- k) <td><%= dentist.dentist_profile.major %></td>
- l) <td><%= dentist.first_name %></td>
- m) <td><%= dentist.last_name %></td>
- n) <td><%= dentist.year_born %></td>
- o) <td><%= link_to 'Destroy', dentist, method: :delete, data: { confirm: 'Are you sure?' } %></td>
- p) <td><%= link_to 'Edit', edit_dentist_path(dentist) %></td>
- q) <td><%= link_to 'Show', dentist %></td>
- r) <th colspan="3"></th>
- s) <th>First name</th>
- t) <th>Last name</th>
- u) <th>Major</th>
- v) <th>Year born</th>
- w) <thead>
- x) <tr>

<u>1)</u>	<u>9)</u>	<u>17)</u>	<u>25)</u>
<u>2)</u>	<u>10)</u>	<u>18)</u>	<u>26)</u>
<u>3)</u>	<u>11)</u>	<u>19)</u>	
<u>4)</u>	<u>12)</u>	<u>20)</u>	
<u>5)</u>	<u>13)</u>	<u>21)</u>	
<u>6)</u>	<u>14)</u>	<u>22)</u>	
<u>7)</u>	<u>15)</u>	<u>23)</u>	
<u>8)</u>	<u>16)</u>	<u>24)</u>	

Solution:

1) h

2) i

3) w

4) x

5) s

6) t

7) v

8) u

9) r

10) g

11) F

12) j

13) a

14) x

15) l

16) m

17) n

18) k

19) q

20) P

21) o

22) g

23) b

24) e

25) d

26) c

Multiple-Choice Questions:

1. Which of the following routes corresponds to the page in Figure 32?
 - a. `get '/dentists', to: 'dentists#index', as: 'dentists'`
 - b. `get '/dentists/:id/edit', to: 'dentists#edit', as: 'edit_dentist'`
 - c. `get '/dentists/:id', to: 'dentists#show', as: 'dentist'`
 - d. `patch '/dentists/:id', to: 'dentists#update'`
 - e. `post '/dentist', to: 'dentists#create'`

2. Which of the following lines of code would the controller need to execute before rendering the view from Figure 32?
 - a. `@dentists = Dentist.all`
 - b. `@dentist = Dentist.new`
 - c. `@dentist = Dentist.new(params.require(:dentist).permit(:first_name, :last_name, :year_born))`
 - d. `@dentist = Dentist.find(params[:id])`
 - e. None of the above

3. True or false? State-affecting controller actions (such as create, update, and destroy) should always send an HTTP redirect response instead of rendering a view.
 - a. True
 - b. False

Solutions:

1. a

2. a

3. a

The questions on the following pages refer to the following example figures. The figures show different aspects of the *beebopdb* web app that is a free and open online music database. Users can use the app to browse and manage data on music artists, albums, and tracks data.

```
# == Schema Information
#
# Table name: artists
#
# id          :integer          not null, primary key
# name       :string
# year_founded :integer
# place_founded :string
# about      :text
# created_at  :datetime         not null
# updated_at  :datetime         not null
#
class Artist < ApplicationRecord
  has_many :albums
  validates :year_founded, numericality: { less_than_or_equal_to: Date.today.year }
end

# == Schema Information
#
# Table name: albums
#
# id          :integer          not null, primary key
# title       :string
# year_released :integer
# genre       :string
# artist_id   :integer
# created_at  :datetime         not null
# updated_at  :datetime         not null
#
# Indexes
#
# index_albums_on_artist_id (artist_id)
#
class Album < ApplicationRecord
  belongs_to :artist
  has_many :tracks
  validates :genre, inclusion: { in: ['Rock', 'R&B/HipHop', 'Pop', 'Country', 'Latin'] }
end

# == Schema Information
#
# Table name: tracks
#
# id          :integer          not null, primary key
# title       :string
# track_number :integer
# length_seconds :integer
# album_id    :integer
# created_at  :datetime         not null
# updated_at  :datetime         not null
#
# Indexes
#
# index_tracks_on_album_id (album_id)
#
class Track < ApplicationRecord
  belongs_to :album
end
```

Figure 33. Three model classes from the beebopdb app.

```

one:
  name: LCD Soundsystem
  year_founded: 2002
  place_founded: Brooklyn
  about: LCD Soundsystem is an American rock band from Brooklyn, New York City...

two:
  name: Arcade Fire
  year_founded: 2001
  place_founded: Montreal
  about: Arcade Fire is a Canadian indie rock band, consisting ...

```

```

one:
  title: This Is Happening
  year_released: 2010
  genre: Rock
  artist: one

two:
  title: The Suburbs
  year_released: 2010
  genre: Rock
  artist: two

```

```

one:
  title: Dance Yrself Clean
  track_number: 1
  length_seconds: 536
  album: one

two:
  title: Ready to Start
  track_number: 2
  length_seconds: 255
  album: two

```

Figure 34. Test fixtures for the beebopdb model classes.

```

(a) end
(b) one.genre = 'INVALID'
(c) test "should be invalid genre" do
(d) one = tracks(:one)
(e) assert one.valid?
(f) test "should be valid artist" do
(g) one.year_founded = Date.today.year + 1
(h) test "should be valid album" do
(i) one = artists(:one)
(j) assert_not one.valid?
(k) test "should be valid track" do
(l) test "should be invalid year founded" do
(m) one = albums(:one)

```

Figure 35. Model unit test lines of code.

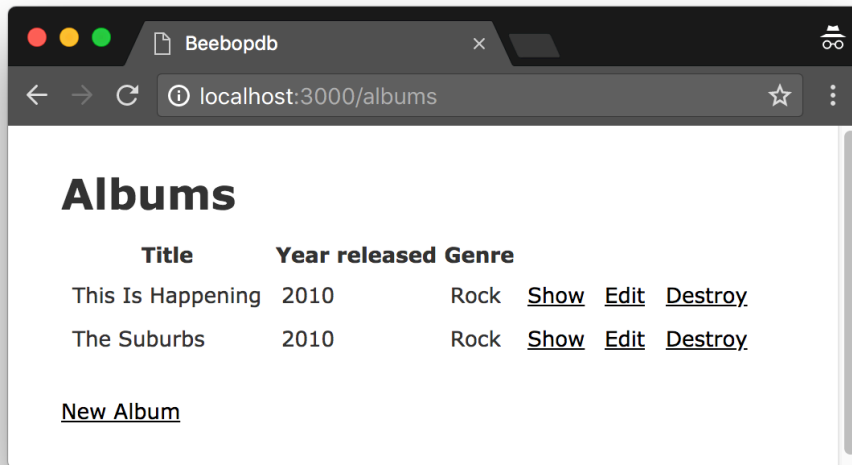


Figure 36. Albums *index* page.

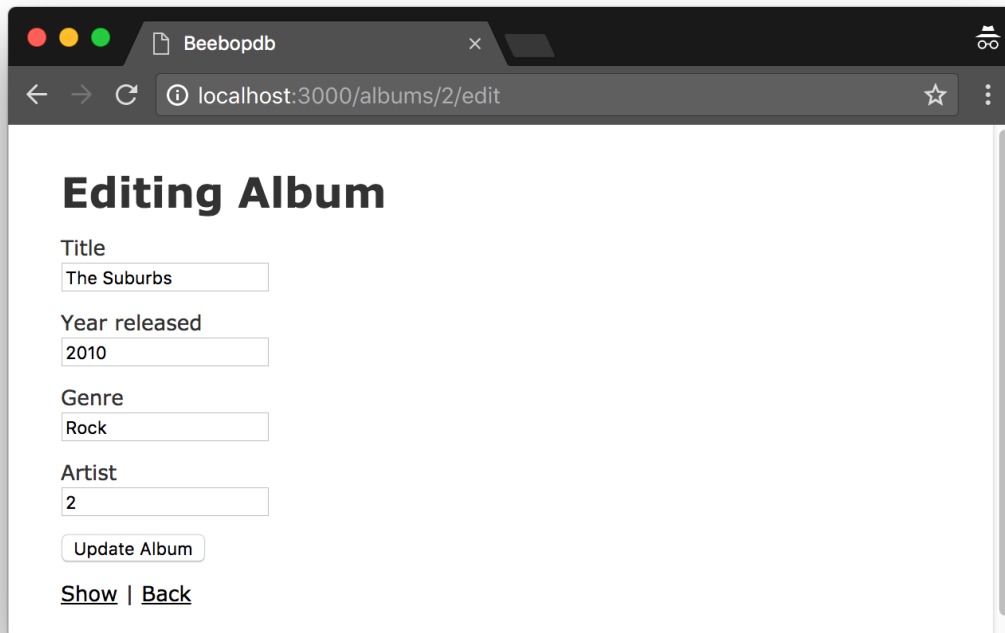


Figure 37. Form for updating an Album.

```

(a) <tbody>
(b) <table>
(c) <td><%= album.year_released %></td>
(d) <% @albums.each do |album| %>
(e) <% end %>
(f) </tbody>
(g) <tr>
(h) <td><%= link_to 'Show', album %></td>
(i) <%= link_to 'New Album', new_album_path %>
(j) </tr>
(k) <td><%= album.genre %></td>
(l) <th>Title</th>
(m) </table>
(n) <td><%= link_to 'Edit', edit_album_path(album) %></td>
(o) <th>Year released</th>
(p) <h1>Albums</h1>
(q) <td><%= link_to 'Destroy', album, method: :delete, data: { confirm: 'Are you sure?'
  } %></td>
(r) <td><%= album.title %></td>
(s) <th colspan="3"></th>
(t) <thead>
(u) </thead>
(v) <th>Genre</th>

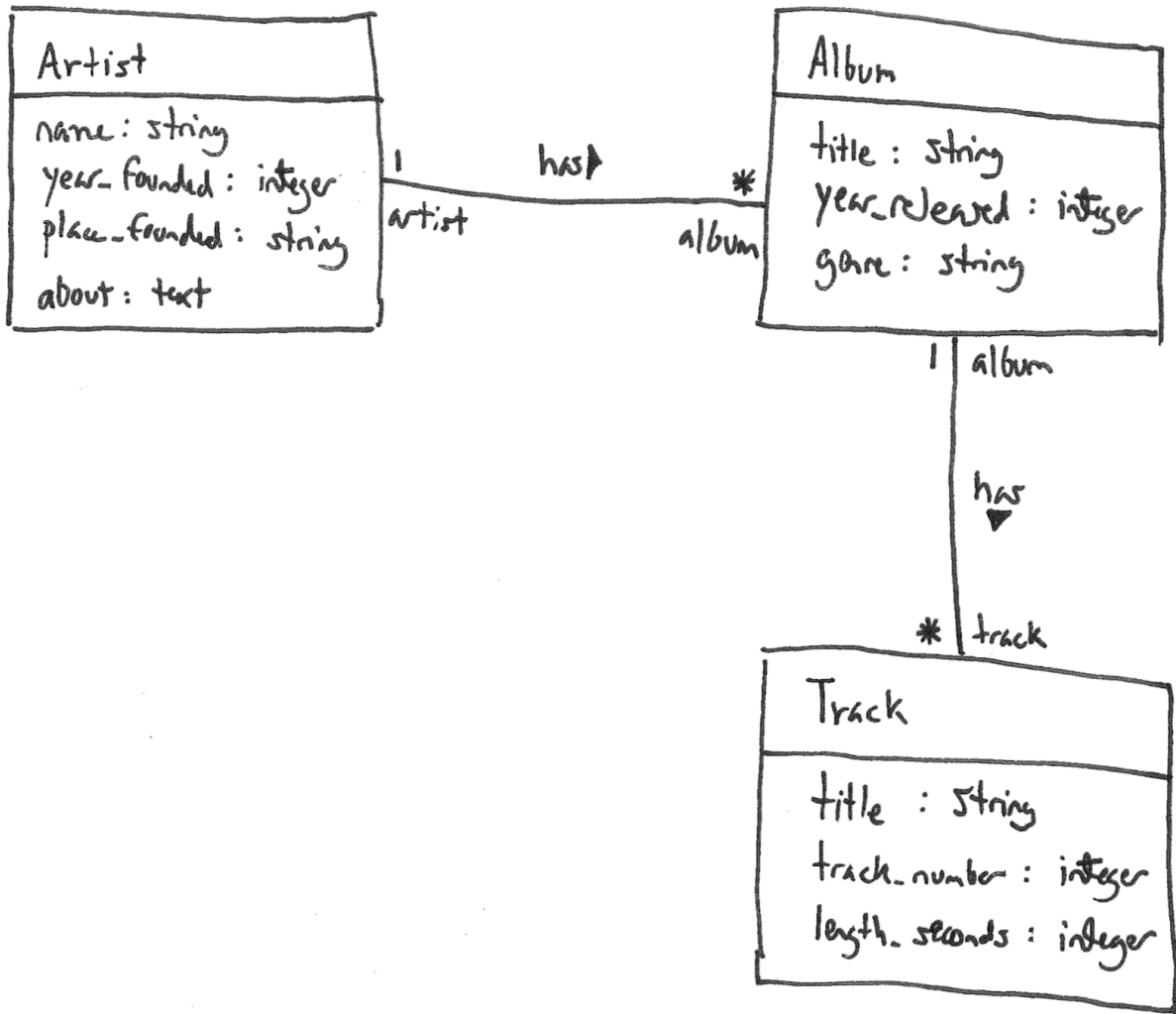
```

Figure 38. Lines of ERB code for the Albums *index* page.

Problem:

Draw a UML class diagram that represents the three model classes given in Figure 30. Be sure to label all associations and association ends, and include all multiplicities. Don't include any "id" attributes (including foreign keys). You may also omit the "datetime" attributes that Rails provides by default.

Solution:



Problem:

Consider the model classes in Figure 30 and the fixtures in Figure 31. Using the lines of code in Figure 35, complete the following model test classes such that each model class has test for a valid instance of the class and such that each validation has a test which demonstrates that the validation catches an invalid value. You should fill all blanks and use all lines at least once. Some lines may be used more than once.

```
class ArtistTest < ActiveSupport::TestCase
```

```
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____
```

```
class AlbumTest < ActiveSupport::TestCase
```

```
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____
```

```
class TrackTest < ActiveSupport::TestCase
```

```
_____  
_____  
_____  
_____  
_____
```

Solution:

```
class ArtistTest < ActiveSupport::TestCase
```

```
  f  
  i  
  e  
  a  
  l  
  i  
  g  
  j  
  a  
  a
```

```
class AlbumTest < ActiveSupport::TestCase
```

```
  h  
  m  
  e  
  a  
  c  
  m  
  b  
  j  
  a  
  a
```

```
class TrackTest < ActiveSupport::TestCase
```

```
  k  
  d  
  e  
  a  
  a
```


Solution:

p
b
g
t
l
o
v
s
j
v
a
d
g
r
c
k
h
n
q
j
e
f
m
i

Problem:

It is possible to add an “Artist” column to the Albums *index* page by inserting two lines of code. What are the two lines of code, and where should they be inserted in your answer to the previous question?

Solution:

First line of code should be inserted after `<th>Genre</th>` (v),
and it should be:

```
<th>Artist</th>
```

Second line of code should be inserted after

```
<td><%= album.genre %></td>
```

 (R), and it should be:

```
<td><%= album.artist.name %></td>
```

Questions:

1. Which of the following routes corresponds to the form in Figure 37?
 - a. `get '/albums/:id', to: 'albums#show', as: 'album'`
 - b. `patch '/albums/:id', to: 'albums#update'`
 - c. `post '/album', to: 'albums#create'`
 - d. `get '/albums/:id/edit', to: 'albums#edit', as: 'edit_album'`
 - e. `get '/albums', to: 'albums#index', as: 'albums'`

2. Which of the following lines of code would the controller need to execute before rendering the form view from Figure 37?
 - a. `@albums = Album.all`
 - b. `@album = Album.new`
 - c. `@album = Album.find(params[:id])`
 - d. `@album = Album.new(params.require(:album).permit(:title, :year_released, :genre, :artist_id))`
 - e. None of the above

3. [1pt] True or false? State-affecting controller actions (such as create, update, and destroy) should always render a view, which produces an HTTP response containing HTML for the browser to display.
 - a. True
 - b. False

Answers:

1. d

2. c

3. b

Problem:

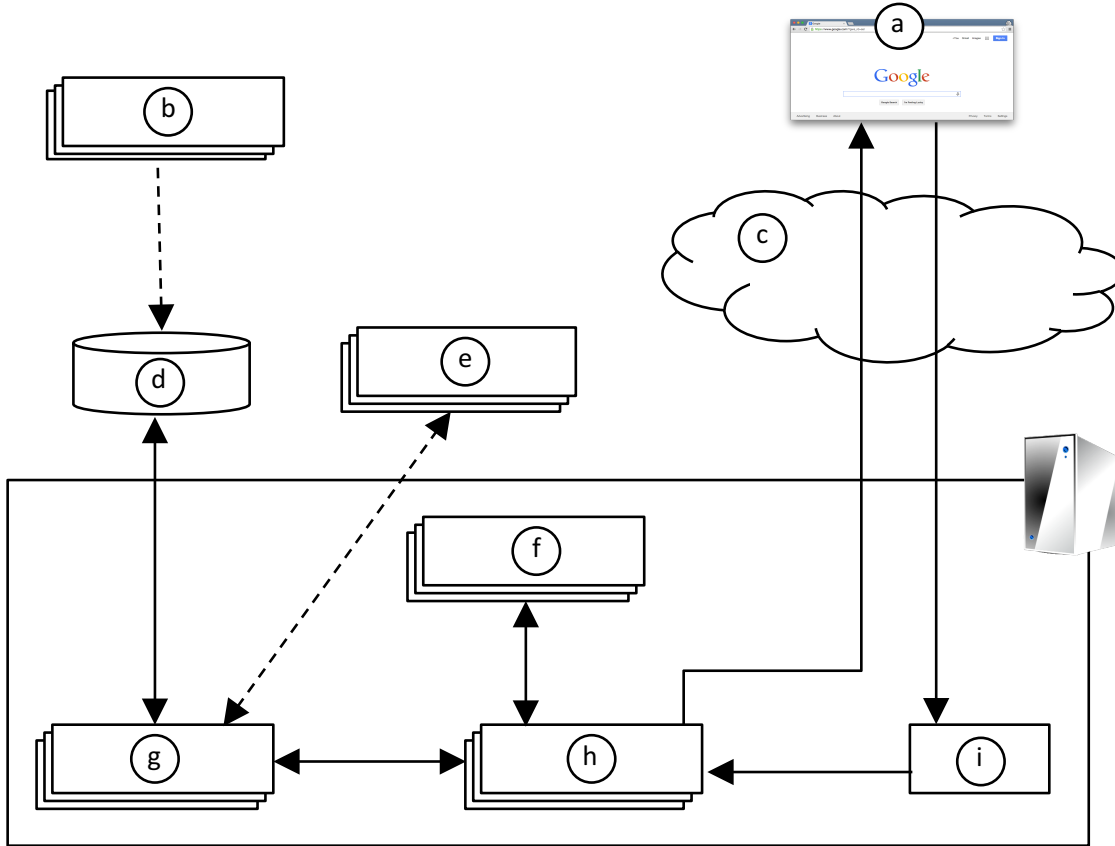


Figure 39. Rails architectural diagram.

For each component below, give the corresponding letter from the Rails architectural diagram in Figure 39.

_____ Model

_____ Browser

_____ Tests

_____ Controller

_____ Migrations

_____ Internet

_____ Database

_____ View

_____ Router

Solution:

g Model

a Browser

e Tests

h Controller

b Migrations

c Internet

d Database

F View

i Router