Eric Burger
IT 232 401
Due 9/23/2013

Final Project Plan: Online Game Store – Digital Distribution

'GameStore'

I. Description

GameStore: An online storefront that allows users to browse and shop from a selection of digital video games.

II. Functional Plan

a. Public Section

Function	Description	Controller	Primary View	Associated Views
Flash/Featured Page	presents featured games and links to public pages	front	Home.html	About.html
List available games, link to purchase	Meant to be a presentable view that displays descriptions, features, and details of game titles	gamelist	Gamelist.html	
User registration and login	Allows the user to create a new	usersession	New.html	User profile page (displays information from Purchasedgame

	account or login with an existing account, and view a profile page that displays purchased games			model)
Game Purchases	Function for purchasing games, associating game products to the user's account	purchase	Purchase.html	
Shopping Cart	Allows the customer to manage a shopping cart before purchasing, link to Purchase	cart	Cart.html	Confirmpurchase.html (simple confirmation that customer is ready to proceed)
Showing Games Purchased by Users	Inputs a username and returns the list of games purchased	Show_purchased_games	Purchase_input.html	Purchase_output.html

b. Maintenance Section

The maintenance section allows access to the databases. Requires an administrator login.

Admins should have CRUD capabilities for each model in this section.

Function	Description	Controller	Primary View	Associated Views
Site Admin login	The process for site administrators to log in for access to the backend	adminsession	New.html	Two views, one for successful login and the other for failure
Site Admin Home	A portal for admins to navigate, presents them with options	adminhome	Adminhome.html (contains links to database views)	_form, edit, index, new, and show for each table

III. Model Plan

Objects: Games, Prices, Users, Purchased Games, Developers, Produced Games, Admins

Planned Models and Schema:

Game – game_id (Int), game_title (String), developer_name (String), publisher (String), price(decimal), release_date (Date), image_name (String)

Price – game_id (Int), price (decimal)

User – user_id (Int), username (String), address (String), city (String), state (String), country (String), zip (Integer), email (string), password_digest (string)

Purchased_game – user_id (Int), game_id(Int), game_title (string)

Developer – dev_id (Int), developer_name (String) games_produced (String)

Produced_game – admin_id (Int), game_title (String), game_id(Int)

Admin – admin_id (Int), admin_name (string), email (string), password_digest (string)

Special Database Tables:
Usersession: user_id (Int), session_id
Adminsession: admin_id (Int), session_id

Associations:
Games – Has_many :developers, Has_one :prices
Prices – belongs_to :games
Users – Has_many :games, :through => purchasedgames
Developers - Has many :games, :through => producedgames

Validators: for the presence of all fields, the data types of integer fields, uniqueness of the following fields: IDs, game_title, developer_name, username, admin_name.

Example validation (app/models/game.rb): validates :game_title, :presence => true, :uniqueness => true

Views

Views related to databases (Game, Price, User, Developer, Admin, etc):
Each model contains the _form, edit, index, new, and show.html.erb views when the scaffold is generated.
Session controllers (usersession and adminsession) will have new.html.erb views for login functionality to their respective access areas. They will also have success and failure views depending on login.

Views related to controllers:
Home.html (flashpage), controller: front
About.html, controller: front
Games.html, controller: games
User.html, controller: users
Purchase.html, controller: purchase

Router Implications
Modify routes.rb:

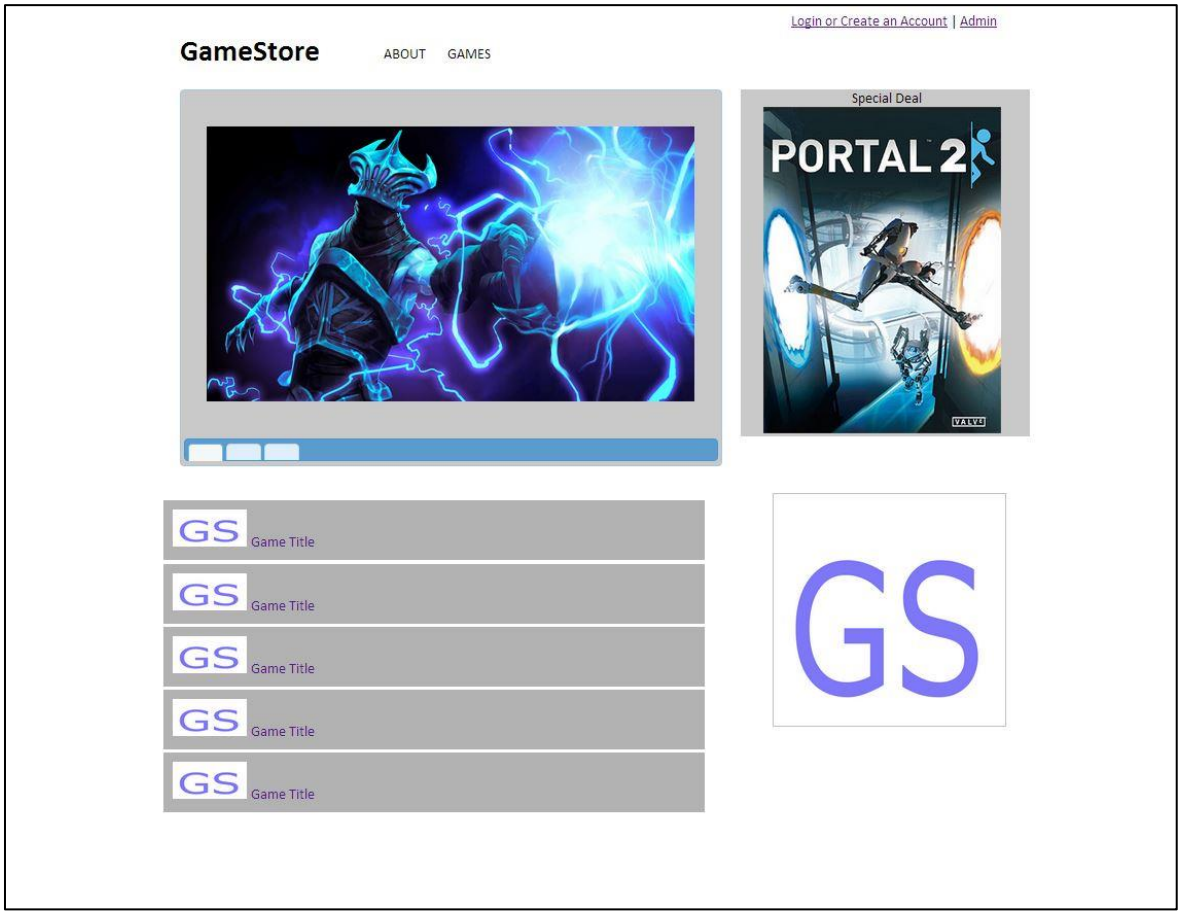
root :to => ‘gamestore#home’ (to set the Flash page)

Delete public/index.html

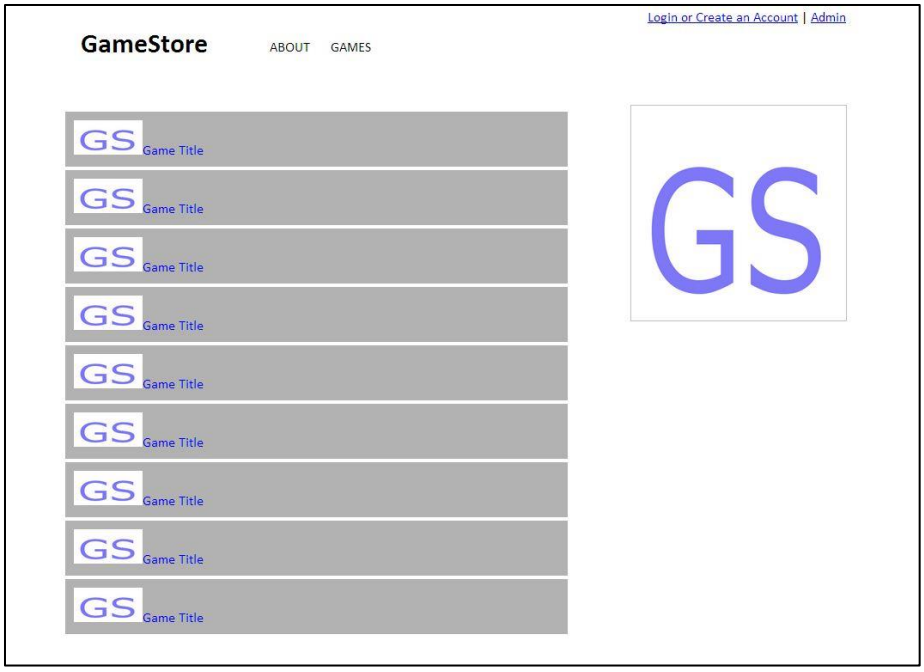
Get views and sessions, get “login” and delete ‘logout’ for adminsessions and usersessions, load resources.

Further Misc. Notes on Development:

Early Mockup of front page:



Early mockup of Games page:



When a user purchases a game, the game becomes associated with the user account.

-Functionality for when a new user is created, a profile view is also generated.

Tentative:

-Do maintenance backend (admin) and public frontend (user) need separate session controllers, or can one session controller handle both sections?

-Each game object having its own public view?

-Adding a new game item to a list on a public view when a game object is added to the Game table?

-Refer to course 9/23 about creating an image column to generate an image thumbnail with each game object.

rails g migration AddImageToCourse image:string --- AddImageToCourse(FunctionAttributenameToClass)

followed by: db:migrate

there is a desc datatype for descriptions, could be used for descriptions instead of strings.

3 & 4 – Flash Page and CSS (Banner)

1.) Generate controller and view for flash page
2.) Build the new CSS
    a. Copy scaffold.css into application.css
    b. Edit the layout application file.
3.) Delete index.html

Model Work

1.) Put validations into model
    a. Edit model
2.) Build a virtual attribute to mine some data not given just by outputting the table
3.) Generate controller and view to output virtual attribute
4.) Links

Linking to the homepage: use root_path

Linking to views in other controllers:

Site Architecture and navigation:

OBJECT ORIENTED DATABASES

Table = Class, Attributes = Methods

Rails g migration AddGame_IdToGame game_id:integer, followed by rake db:migrate

Rails g migration AddUser_IdToUser user_id:integer

To Do:

Front end page for maintenance section

Links

Images

Remove game_id and user_id attributes

Associations – Requires associating our database tables or classes hooking them together

1. One to one
   Customer -> account
   Course -> price
2. One to many
   Customer -> courses

   Tasks to create an association:
   (one to one, one to many, many to many)
   2 tables, one of which has a foreign key (rails g migration Addto…)
   Ex: customer, courses, Customer table can have Course ID or courses can have a customer ID
   Populate the tables
   Edit the models with the association statements(has_, belongs)
   Create a controller for the association

   Verification:
   1.) New data Class
   2.) Populate
   3.) Build controller
   4.) Build the views

   1.) Users table (class)
   -name
   -password_digest (encrypting): String
   -password: String
   -password_confirmation: String

   2.) controller Sessions new

   Then edit the controller

Populate the user table

Build your new input page

Take in username and password.

Two sections to coding portion for this assignment:

I. Build the basic authentication
II. Protect the system with filters to determine which pages are protected by login
    a. Put filter code into the application controller
    b. Put in the skip before filter code in the public controllers
        “skip_before_filter :authorize”
        Also goes into the Sessions controller, otherwise the user cannot see the login page