|  |  |  |
| --- | --- | --- |
| **ExploreMore - Software Development Team** | | |
|  |  |  |
| **Fintan Moloney** | **James Bentley** | **Niall Conway** |
| Project Manager, sourcing new repositories, setup of the environment. Meeting management and task distribution. | Technical Lead - implementation of the code, reporting back to team and walkthrough of progress and issues encounters - to share learnings. | Documentation Lead - to ensure requirements met and detailed. Technical input incl. implementation of non-core gem extensions and repos (e.g. Cookie Consent). |

**Project Description**

Design and develop an e-commerce based web application for the domain/business.

* Users log in & browse inventory by attribute e.g. product name, manufacturer, model etc.
* User register with your site to make allow for purchases on the site
* can then add items to cart.
* can be added and removed by the user as browses the inventory.
* When ready to make a purchase the user is brought to a payment page
* details on what is in the user’s cart shown to user and total price is also provided.
* User makes a payment for the purchases.

Responsive web design (RWD) in order to provide optimal viewing on client-side multidevice.

**Project Description**

1. Describe domain (i.e. field of business) e-commerce app developed to serve. incl.

* conceptual description of domain identifying core concepts/entities (info held on things).

2. Specific requirements of domain: functions, tasks, processes etc. (equiv of project scope doc)

3. Proposed web application specification

* Features in web app (to level of detail required to begin implementation)
* how user navigates app (diagram on web app structure, workflow invl for core functions.

4. Proposal mock-ups of key screens.

**TECHNICAL SPECIFICATION REPORT**

**Introduction** - brief review of proposal, any new additions since the proposal was submitted.

**Model** eg. example code entities, key entity attributes, primary/foreign keys, assoc relationships

**Key Users Flows** - Routing of your system. List everything a user can do and for each action identify the URL, controller and method/action which that user action calls. Description of what the controller does (e.g. loads in all items, loads item with id etc.) and then identify the view returned, again with a brief description of what happens during the creation of the view (e.g. items from controller used to create a list of items). Incl. both customer and admin flows. Identify any session or flash variables used during these user flows (e.g. like cart code in Golf example).

**Access control** - List levels of users (e.g. admin, normal user) and any access restrictions

based on the user level. Use this data when we implement the devise login gem later.

**Any 3rd party gems** - intend to use, give a brief description of what their function is (not required)

**Any external (3rd Party) interfaces** - intend to communicate with (e.g. a payment gateway etc.) and briefly describe the technical interface (not required).

**Required Elements**

1. Web application will be implemented using the Rails framework. Site implements
2. Users register with site to buy items.
3. Database manages user details.
4. Unregistered users can view content but cannot buy items.
5. Users can buy more than one item in a given session.
6. Log in area provided on all screens for a user who is not logged in.
7. Number of products user has put into “cart” and subtotal displayed to user at all times.
8. When user has bought an item - must be recorded in database.
9. Website is secure by:
   1. Taking measures against A. SQL/HTML injection and B. redirection. (see below)
   2. Proper session management. (authentication)

|  |
| --- |
| Sessions enable the application to maintain user-specific state, while users interact with the application. For example, sessions allow users to authenticate once and remain signed in for future requests. A session is a place to store data during one request that you can read during later requests. You can set some data in a controller action and read it in another: **app/controllers/sessions\_controller.rb**  When you request a webpage, the server can set a cookie when it responds back. Your browser will store those cookies. And until the cookie expires, every time you make a request, your browser will send the cookies back to the server. Your Rails app is in charge of figuring out what a cookie means.  *Rails uses ActionDispatch::Session::CookieStore as the default session storage.* |

* 1. User credentials are protected at database level using encryption.

|  |
| --- |
| This is the answer - config/credentials.yml.enc |

**Website - minimal number of pages:**

* **Welcome** - the user, provides user with info on last visit and what they bought.
* **Profile** - user details, can be edited by user. User’s purchase history.
* **Displays Inventory** - can be browsed in variety of ways and can be searched.
* **Registration** - where users log into page
* **Check out** - # of each item being bought & total price. Allow user to add items to a cart.
* **Thank you** - after a purchase is made.

**Examination Method**

Present web application to demonstrate understanding of implemented solution.

1500 word technical specification report describes:

* Implementation
* issues encountered/overcome
* decisions made during implementation
* explanation of any proposed features not implemented.
* Examples or tutorials to be detailed.

Final code submission

* incl Heroku (or similar) URL with a live copy of application running
* public bitbucket (or similar git online repository) URL for the code repository.
* Code submitted via zip file on Moodle.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CRITERIA (this is where we get the points)**

**1. Web Technology:**

Effectively address:

* session management
* security issues
* designed using software engineering principle.

Understand MVC architecture and its application in Rails.

* Extension to ActiveRecord, controllers and views for advanced functionality.
* DRY principle is effectively applied.

|  |  |
| --- | --- |
| **ExploreMore - Model View Controller** | |
|  | 1. User interacts with a view. 2. View alerts controller of particular event. 3. Contoller updates model. 4. Model alerts view that it has changed. 5. View grabs model data and updates itself. |

**Data Persistence:**

* ActiveRecord extend to incorporate additional functionality.

|  |
| --- |
| **Active Record**  - The M in MVC - the model (i.e. the layer of the system responsible for representing business data & logic.  - Facilitates the creation and use of business objects whose data requires persistent storage to a database.  - An implementation of the Active Record pattern - a description of an Object Relational Mapping system.  **Another Description**  Architectural pattern found in software that stores in-memory object data in relational databases.  The interface of this pattern includes:  - functions such as Insert, Update, and Delete  - properties corresponding directly to the columns in the underlying database table.  An approach to accessing data in a database.  - database table or view is wrapped into a class.  - an object instance is tied to a single row in the table.  - After creation of an object, a new row is added to the table upon save.  - Any object loaded gets its information from the database.  - When an object is updated, the corresponding row in the table is also updated.  - The wrapper class implements accessor methods or properties for each column in the table or view.  This pattern is commonly used by object persistence tools and in object-relational mapping (ORM). Typically, foreign key relationships will be exposed as an object instance of the appropriate type via a property.  **Active Record Migrations**  Migrations are a feature of Active Record that allows you to evolve your database schema over time. Rather than write schema modifications in pure SQL, migrations allow you to use a Ruby DSL to describe changes to your tables. |

**Completeness:**

Develop using sound software engineering principles.

|  |
| --- |
| 1. KISS (Keep It Simple, Stupid!)  2. DRY (Don’t Repeat Yourself)  3. YAGNI (You Aren’t Gonna Need It)  4. S.O.L.I.D. Principles  - Single Responsibility - every module or class should have only a single responsibility (i.e. only changes to one part of the software’s specification should be able to affect the specification of the class).  - Open/Closed - software entities should be open for extension but closed for modification.  - Liskov Substitution - objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.  - Interface Segregation - many client-specific interfaces are better than one general-purpose interface.  - Dependency Inversion - depend upon abstractions, [not] concretions. |

Carries out all above functionality.

Provide user with other unconsidered functionality or non-functional requirement e.g.

* additional security provisions made
* recommendations made to the user based on data from database.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **Iconography to include in Documentation (representing):** | |
| People (Roles and users) |  |
| Database (Model) |  |
| Paper document (Manual, documents) |  |
| View |  |
| Information (Data to pass through app) |  |
| Procedures |  |
| Procedure (MVC etc) |  |
| Gem (Installed for app) |  |
| Web Online |  |
| Mouse pointer and button (Functionality/Controller) |  |
| Computer (Devices) |  |
| Technical |  |
| Certification |  |

**Moodle Notes to Reference**

* Static pages
* App-add products
* Styling the form
* Basic cart and basic login (create limited user functionality, write cart in ruby)
* Finishing cart functions (continue shopping, delete items from cart)
* Authentication with device
* Processing orders and filtering orders per customers
* Extra functionality
  + Create category
  + Search in Rails
  + Paypal example
  + Popular products
  + Admin in devise

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **RESOURCES** | |
|  | Ruby is an interpreted, high-level, general-purpose programming language. |
|  | Ruby on Rails is a server-side web application framework written in Ruby under the MIT License. Rails is a model–view–controller framework, providing default structures for a database, a web service, and web pages. |
|  | Hypertext Markup Language is the standard markup language for documents designed to be displayed in a web browser. |
|  | JavaScript is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. |
|  | Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language like HTML. |
|  | **AWS Cloud9** is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. |
|  | Cloud9 IDE is an Online IDE, published as open source from version 2.0, until version 3.0. It supports multiple programming languages, including C, C++, PHP, Ruby, Perl, Python, JavaScript with Node.js, and Go. It is written almost entirely in JavaScript, and uses Node.js on the back-end. |
|  | Github version control software – access will allow for a more accurate assessment of each individual – incl details of repository used for project.  <https://github.com/Josebuendia/ExploreMoreRailsApp> |
|  | Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to deploy, manage, and scale modern apps. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| **Notes on:****SQL Injection Prevention Techniques for Ruby on Rails Web Apps**  1. **Use Dynamic Attribute-Based Finders When Possible**   Example:  User.find\_by(name: params[:name]) # Traditional  User.find\_by\_name(name) # dynamic finder  These work as parameterized queries and take care of the passed argument. Protection - passing the column name and value explicitly. Dynamic-attribute based finders, didn't really require anything rather than calling the method itself. Active Record knows that the argument is not a column or a table name and added to the SQL query with proper escaping.   1. **Only Accept and Construct Values From Inputs, not Commands**   e.g. INSERT, DELETE, and SELECT and other core parts of SQL queries such as column and table names should never be constructed from untrusted inputs. Can be easily achieved by not sending untrusted inputs to those methods in the Active Record library that accept column and table names or raw SQL queries.   1. **Never Pass a String as an Argument Unless It Is a Dynamic Finder**   Must know exactly what the AR method expects as arguments. Some methods accept arguments in multiple forms and accept different formats. Method that accepts integer will be vulnerable to SQL Injection when called with a string.  Many methods usually require external input to work within the Active Record, esp. under **ActiveRecord::FinderMethods** and **ActiveRecord::QueryMethods**. User input should not be passed as it is - instead encapsulated in a form that prevents modifying the SQL query.  Example: **ActiveRecord::QueryMethods::WhereChain::where** method  - does the filtering based on the conditions specified in the arguments.  - method accepts conditions in several different formats such as string, array, or hash, and goes to the WHERE-part of a SQL statement.  - If passing user\_input directly to this method: would create a very basic type of SQL Injection because when a string is passed, it will be added to the query as a SQL fragment and providing such an attack opportunity.  **Vulnerable versions:**  User.where("name = '#{params[:name]'") # SQL Injection!  If this particular line of code invoked with name = 'fff', the resulting query will be:  SELECT "users".\* FROM "users" WHERE (name = 'fff')  => #<ActiveRecord::Relation []>  But if it is set to "' **OR 1='1**":  SELECT "users".\* FROM "users" WHERE (name = ' ' OR '1'='1')  => #<ActiveRecord::Relation [#<User id: 1, name:'jack', .......>]>  Reason: successful inclusion of an OR operator which helped us return all the records from the database.  **Injection-proof versions.**  User.where(["name = ?", "#{params[:name]}"])  Reason: the first element of the array is a template and the latter are parameters to that template.  User.where({ name: params[:name] })  Reason: the column name is set explicitly to the 'name' and the external input set to the value of it. |

|  |
| --- |
| **Userful Graphics from Original Website for Report and New Website** |
|  |