AutoRSpec

Dan Shreeve

2017-04-25

Signed declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Daniel Demaine Shreeve

Signature: ....

Date: 03/05/2017

Abstract

Software Testing benefits the development and maintenance of a system by increasing quality and reliability. Software Testing also contributes around half of the cost of producing a system. Automating part or whole of this process reduces the cost whilst maintaining the benefits. The aim of this project is to produce a system that automatically generates RSpec test cases for model validation in Ruby on Rails. The tests will be generated from a formal database specification that the user has defined using the system. A file can be generated and inserted into the user's application and run as though they had been written by the user. The time taken to insert the necessary information should be less than the time taken to write the tests manually, otherwise the user does not benefit.

Acknowledgements

First and foremost I would like to thank my supervisor **Dr. Gordon Fraser** for accepting my proposed project and providing impeccable advice and guidance throughout the process.

Finally I would like to thank my parents, **Linda Shreeve** and **Paul Shreeve**, and my grandparents **Elsie Marsden** and **Ray Marsden** for giving me the oppurtunity to go to university and supporting me throughout.

Contents

Title page i

Signed declaration ii

Abstract iii

Acknowledgements iv

1 Chapter 1: Introduction 1

2 Chapter 2: Literature Review and Research 3
2.1 Testing and Automation . . . 3
2.2 Testing in Rails . . . 7
2.3 Tools to use . . . 8
2.4 Evaluation . . . 11

3 Chapter 3: Requirements and Analysis 13

4 Chapter 4: Design 17
4.1 Database . . . 17
4.2 View and Flow . . . 19
4.3 Test Case Generation . . . 23
4.3.1 Value Generation . . . 23
4.3.2 RSpec Test Case . . . 24
4.3.3 RSpec Test Suite . . . 25

5 Chapter 5: Implemention and Testing 26
5.1 Setup . . . 26
5.2 Database . . . 27
5.3 View and Flow . . . 27
5.4 Test Case Generation . . . 28
5.4.1 Value Generation . . . 28
5.4.2 RSpec Test Case . . . 29
5.4.3 RSpec Test Suite . . . 29

6 Chapter 6: Results and Discussion 29
6.1 Evaluation . . . 29
6.2 Future . . . 30
6.2.1 Improvements to current system . . . 30
6.2.2 System additions . . . 30

6.2.3 Functional Specification 31

7 Chapter 7: Conclusions **31**

1 Chapter 1: Introduction

How severe can the consequences be from an error in a piece of software? In 1983 a bug in a piece of software almost started World War Three.

During the Cold War, tensions between the US and Soviet Russia were extremely high. A Soviet early warning system had detected the launch of five ballistic missiles from the US. The only reason that Soviet Russia did not retaliate, thereby starting World War Three, was the fact that Lt Col Stanislav Petrov had a "...funny feeling in my gut"[1] and concluded that if the US was launching a full scale attack they would launch more than five missiles. The error in the system was discovered to be a bug in part of the software that distinguished false missiles from satellites picking up the reflection of sunlight from the top of clouds.[1] If the bug in the code had falslely detected more missiles the world could be a very different place today.

Software testing increases the relability of a system and quality, by detecting errors and bugs which can be fixed, quality can aslo be further improved by tests proving that the system meets its design requirements. However, software testing is very costly, accounting for half the time put into development of a system and half of total expenditure.[2] By automating part or all of the software testing process, testing costs can be reduced, whilst maintaining the benefits.

The motivation for this project is to bring the benefits of automated testing to Ruby on Rails. Ruby on Rails is a web application framework that allows developers to create fully functioning applications in a short of space of time. The reduction in time spent implementing increases the proportion of time spent testing. Thefore Ruby on Rails would benefit greatly from automated testing. This would further enhance the time saving feature of Ruby on Rails.

This project aims to create a system that Ruby on Rails developers can use to reduce the development time of their projects by automatically generating all or part of the tests they require. The tests generated should not reduce the reliability and quality that manually written tests create.

WHATS TO FOLLOW GENERAL

WHAT CHP 2 IS ABOUT

WHAT CHP 3 IS ABOUT

WHAT CHP 4 IS ABOUT

WHAT CHP 5 IS ABOUT

WHAT CHP 6 IS ABOUT

WHAT CHP 7 IS ABOUT

2 Chapter 2: Literature Review and Research

2.1 Testing and Automation

The European Space Agency spent ten years and $7 billion designing and constructing the Ariane 5, a rocket that can launch multiple satellites into orbit from a single launch. Thirty nine seconds into its maiden voyage it exploded, destroying the Ariane 5 and its cargo of four uninsured, extremeley expensive scientific satellites. The explosion was caused by its own self-destruct sequence which was triggered automatically as the boosters were being torn away by aerodynamic forces. These extreme forces were caused by the rocket trying to recorrect its course in response to flight data provided by the guidance system. The guidance system, along with its backup, had crashed and shutdown. The flight data provided, that caused the rocket to readjust its course, was actually a diagnostic error message.

The cause of the shutdown was the guidance system trying to convert the sideways velocity of the rocket from 64-bit to 16-bit. To make matters worse, the programmers were aware that it could overflow but assumed that the particular causal variable would never be large enough, as it was used to prepare for launch and not in flight. However it was decided the system should run into the first forty seconds of flight, in case of a brief hold in the launch countdown, to make restarting the system easier. A known flaw in a system, that could have been handled, resulted in a chain of events that led to the catastrophic explosion of Ariane 5.[3].

Software disasters can be caused by poor testing practices[4], if the correct testing procedures and practices had been in place for both Ariane 5 and the Soviet Guidance System example from the Introduction these situations could have been avoided. Software testing is therefore extremely important and should be included in the development of all software applications. What is software testing and how does it help avoid these situations ?

Software testing is an investigation into a piece of software that provides information during development and maintenance. A process or series of processes are carried out that are designed to make sure computer code does what it designed to do and is absent of unintended behaviour[2]. The information retrieved from the proccesses can be used to track the progress during development against acceptance criteria and detect and locate errors and bugs. Errors and bugs detected within the code of the are immediatley known and can be handled, providing a smoother and more consistent development and maintenance flow. Software testing provides a more reliable and higher quality product when used as part of the development process due to these benefits.

Testing however can not guarantee that a program or peice of code is without errors, therefor completing testing is impossible. This is why the design of tests is vital to the integrity of the testing, making the tests as complete as possible. Given the constraints on time and cost, effective testing is simply "What subset of all possible test cases has the highest probability of detecting the most errors?"[2]. Tests are designed using information about the program along with its intended behaviour. In a given environment, with proper determined input, there is an expected behaviour/output. If the code under test does not display the desired outcome it is said to have failed the test.

A 'test case' will test a very specific behaviour of a program. A collection of test cases is a 'test suite', representing that a certain section of the system has a specificied set of behavoiurs. A relevant example would be for a table in a database. The test suite would represent if the table has the desired validations in place and would consist of test cases that tested each specific behaviour in isolation. The tests would be run as a set to confirm the table has the desired behaviour. When undesired behaviour is detected the test result will highlight the exact error in the code. The design of these test cases is crucial. To design these cases information is required, this information is sourced in two main ways Functional and Structural.

Functional Testing also known Black-box testing, is the technique of creating test cases with information from a formal or informal functional specification. A functional specification is the description of intended program behaviour distinct from the program itself. The software requirements and or its design specification are most commonly used to derive the functional specification. The software entity under test is treated as a black box, the actual code implementation is not known, where proper inputs are fed in and the output is observed. If the output or behaviour is that specified in the specification the test has been passed, otherwise it has been failed. Example: When a user clicks the Home tab in the nav bar they are directed to the Home page, in this case the input is the user clicking the home tab and the desired behaviour is being directed to the home page. Black box testing can detect some faults that white box cannot, such as absent behaviours that are in the functional specification of requirements but not coded into the software application in error. The systematic nature of Black-box testing can help avoid missed test cases and provide more consistent coverage.[5][6]

One approach to functional testing is a systematic approach. A systematic approach has four steps :[6]

1. Partition the functional specification into independently testable features using a divide and conquer approach. For example, for a database table, dividing a table in a databse into its fields, then dividing again into the properties of each field.
2. For each independently testable feature find a representative class of values or derive a model to test it. For a string field with property of length greater than five. We may derive a specail case of blank string, a string of length less than five and string with length greater than five.
3. Generate test case specifications. Finding concrete values for the reprensentive class of values or model above. Building on the previous example we may have { "", "less" ,"longer"}
4. Generate test cases and institute tests, turning specifications into tests and instituting them.

Structural testing, also known as White box testing. uses the physical implementation of the software itself such as source code as information to produce test cases. A common approach to white box testing is 'Control Flow' testing. Due to the nature of code varying greatly between projects and the complexity and detail that can be involved the following steps have been simplified whilst maintaining the core principle.

1. Identify a feature to be tested. This could be on a small or large scale, for our examples we will choose creating an entry on a database.
2. Create a flow graph plotting all steps and paths the feature can execute, this would include steps such as verification.
3. Identify paths through the flow graph, entry is created successfully and

displayed or unsuccessfuly and prompted with an error message.

4. For each path write a test case that is expected to execute this path, with valid variables an entry should be created and it should be displayed.

Software testing is necessary and very costly. "In a typical programming project approximately 50 percent of the elapsed time and more than 50 percent of the total cost were expended in testing the program or system being developed"[2]. Reducing the costs, both time and monetary, is the main motivation for Automated Testing. Another overlooked and unappreciated benefit of automating testing is that test case generation is one of the most intellectully demanding and critical challenges in software testing.[7] By automating this process it not only reduces costs but also allows developers to dedicate more time and effort to other areas, also in some cases it is harder to create a test case but easy to verify a generated test case is correct. A whole systems tests do not have to be automatically generated to reduce costs and benefit.

An Automated functional testing approach follows the same methodology as manual systematic generation of test cases. Each step is automated and follows the same principles. The information as before used to derive test cases is the functional specification. The functional specification is however only formal and has specified syntax so that it can be interpreted by the system that will generate the test cases. There is more creativitivy and design put into the functional specification as the test designer is usually limited to a choice of test selection criteria. For step 3 the system must also include value generation that can meet the representative class of values outlined in step 2.[6]

Automated functional testing tends to be more complex, as it has to understand and interpret human written code. One method that has recieved alot of attention from researchers and is Symbolic Execution. This approach is similar to that of Control Flow, where symbolic values, instead of concrete values, are used instead for program inputs. Programs variables are described by the symbolic expressions of those inputs. The state of the program includes the symbolic values of program variables, a program counter and the path contraint on symbolic values: a boolean formula over the symbolic values input. Using this method it can explore all possible path divergences through a system and identify stop points, where the path ends. The major problem with automated White-Box testing is identifying if a behaviour, stop point or a specific divergence in Symbolic execution, is desired or not. This problem is known as the Oracle problem, as desired behavoiur of code is contained within its specfication and design, not its implementation. Therefore some level of user input is required.[7]

Another relevant challenge for Symbolic Execution is developing a system that can cover multiple languages at once. This is very complex and producing a system can produce feasible output can be impossible due to the path divergence problem, where, either a user has to specify so many models automation is not feasible, or it doesnt find a signifcant amount of feasible program paths. As Ruby on Rails design environment can vary drastically between projects due to the flexibility of its framwork and use of Gems I will only consider and that some level of User input is required I will only consider Black Box testing techniques when I come to designing the project. This will deliver a product that will be usable to a wider audience as it is dependent upon on specification for which I can define. Interpreting multiple languages and being flexible enough to be useful is out of the scope of this project.

PARAGRAPH SAYING BB CHOSEN SO VALUE GENERATION NEEDS TO BE CONSIDERED WHEN CHOOSING TOOLS

2.2 Testing in Rails

Ruby on Rails applications are primarily developed using the generation of skeletons with variables set by the developer. The skeletons save a vast amount of time by generating a default environment which can then be built upon according to the section under development. By default one of the files created is a test file and it uses the Ruby on Rails MiniTest class.[8]

TEST UNIT

RSPEC

CONCLUDE RSPEC

RSPEC IN GREATER DETAIL (may be few par)

TEMPLATES FOR RSPEC - NEED STRING MANIPULATION ETC

FACTORY GIRL

2.3 Tools to use

TOOL REQUIREMENTS FROM PREVISOUS SECTIONS

An MVC web application fits all these criteria. MVC, Model-View-Controller is an architectural pattern that seperates an application into three interconnected parts. This separation (of the three components) allows for responsibilities to be allocated independently to each component, seperating the logic from the user. The model is responsible for the data of the application and the rules and logic used to create and update the information. The view is responsible for displaying the data and possible interactions with the system to the user. The controller is responsible for controlling the flow of the system, accepting user input and converting it into commands for the model and view.

An example of the components interacting would be creating an entry to a database. The "view" would be responsible for displaying the form in which to fill in. On submission the "controller" will process the information, ensure only persmissable information is submitted and enter additional information, then send it to the "model". The model will verify the structure of the information, ensure the correct fields are present and accord with its rules. The model will then notify the controller if the submission was successful or not and the controller will update the view to reflect the status.

This seperation of components means that all user interaction with the database has to go through the controller and is therefore limited to what the developers want users to be able to do. This provides a high level of security as each action is controlled and the internal structure and representation of the information within the database is hidden. Simultaneous development is also possible due to the seperation of the components, work on the front and back end concurrently. Although I will not be able to get the full benefit of this aspect (as I am developing the project solo), it will allow me to shift focus as components do not need to be finished when switching from one to another, giving greater flexibility in development.

High levels of cohesion are inherited automatically from the architecture with the grouping of logically similar elements, this makes the code easier to read and creates a more natural flow within the source code. There are however some drawbacks to MVC architecture, they are inherently more complex due to the seperation and the framework must be learned. Also there may be multiple programming languages involved. This steep learning curve could mean a large initial investment (time/cost) into a development team to learn a new framework and new languages.

MVC web application frameworks have become extremely popular and are behind some of the most used and powerful websites. Django an MTV, follows MVC architecture but its creators decided to rename the components [9] to better suit them, is behind the two most visted websites in the world Google and YouTube[10][11]. Ruby on Rails another MVC is behind Twitter, Airbnb and Soundcloud.[12] MVC frameworks are known for their scalablity, being suitable for the smallest to the largest projects. However FaceBook decided that its scalablity had reached its limit, that adding new features made the code exponentailly more complex.[13] My project will be no where near the scale of facebooks sourcecode so I do not need to worry about reaching the end of its scalibilty.

The chosen MVC to construct the project in is Ruby on Rails. I have done previous projects in both Ruby and the Rails framework, the rest of this section will show that Ruby on Rails is an adequate choice.

Ruby was selected as the primary programming language by default as it is the language that runs Ruby On Rails. Ruby is a dynamic, multi-paradigm programming language. The paradigms consist of Object-oriented, Imperative, Functional and Reflective making it a very powerful and versatile language. This combination is from its founder Yukihiro Matsumoto who was influenced by Perl, Smalltalk, Eiffel, Ada, and Lisp.

Rubys primary design goal was to make a language that he himself enjoyed using, by minimizing programmer work and the potential for confusion(Ruby Wiki). Achieved with a focus on human interaction, how programmers code and design applications as opposed to focusing on how the code will run on machines. And also following the principle of least astonishment, where the behaviour of the language minimizes confusion for experienced users.

The above two images show C++ 1 and Ruby 2 printing Hello world to the console. The comparison between the two languages highlights the efficiency and simplicity of the Ruby language.Ruby on Rails projects are commonly

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Figure 1: C++ print Hello world to console

```ruby
puts "hello world"
```

Figure 2: Ruby print Hello world to console

worked on by a group of people and in multiple languages, therefore the simplistic syntax gives greater clarity and understandability to programmers who are lesser experienced in Ruby.[14]

Ruby is open source, free and redistributable with a vast range of existing code from both Ruby and its large community. Primarily consisting of Gems, code packages that can be installed and supported into a project easily and with minimal effort via RubyGems, and frameworks, such as Ruby on Rails. Making it very popular for education and business. Following the DRY Don't Repeat Yourself principle in a very effective and efficient manner.[14]

Ruby was ranked ninth on TIOBE index[**?**] and has become a very popular and respected language relative to its age among the other languages on the index. No alternatives could be considered due to the dependency of Ruby on Rails on Ruby, however Ruby is a very strong and durable language so it does not detract from the overall project.

FORMS AND DATABASE SUPPORT FOR INPUTTING INFORMATION THAT CAN BE PROCESSED

STRING MANIPULATION FOR TEMPLATES

NUMBER GENERATION

STRING GENERATION FROM REGEXP USING GEM REGEXP.RANDOM

RUBY HAS NECESSARY LOGIC TO CONTROL FLOW ETC

2.4 Evaluation

HOW TESTS ARE EVALUATED

HOW OTHER PEOPLE EVALUATED PROJECTS

INJECTION

DOG FEED

COMPARE OUTPUT WITH WHAT EXISTING HAD

COMPARE CODE COVERAGE

TIME SAVED

3 Chapter 3: Requirements and Analysis

REFINE AIM INLINE WITH LIT REVIEW TO "TO PROVIDE A SYSTEM THAT AUTOMATICALLY GENERATES MODEL TESTS IN RSPEC"

This project started with the motivation of bringing automated testing to Ruby on Rails. From research carried out and disscussed in Chapter 2 and considering the scale of the project the way to achieve this was to focus on validation in the model component and produce RSpec test cases. A more precise description of the project is therefore **To create a system that automatically generates RSpec test cases for model validation in Ruby on Rails Applications**. To accomplish this aim there are three main aspects to the system, as follows:

1. Allow a user to create and maintain a formal specifcation

   A user should be able to describe all tables, fields and corresponding validation properties using the system. The user should also be able to edit and update the formal specification due to their changing demands.

2. Generate a valid RSpec test case

   The RSpec test case should be generated with information entered from defined formal specification. To be valid the RSpec test case must meet certain critera. The test case must run the same as a manually produced user test case. It must only test the specified behaviour, e.g Employee table, Age field, must be greater than 18, and nothing else. It must produce a human readable test case with a human readable behaviour descriptor that makes the user aware of exactly what behaviour has failed. It must produce a generated value that isolates the behaviour under test, it generates a value that fails the validation under test while passing the other validations for the field.

3. Consolidate all RSpec test cases for to make a valid test suite for a table

   All test cases for a table, for all its fields and assigned validations, must be consolidated into a test suite that runs as a manual written test suite.

To make the system viable and useful to developers there are some additional considerations

1. Ease of use

The system built should be easy and intuitive to use.

2. Time saving
3. Creating the formal specification must take less time than it takes to write the tests manually

Ruby on Rails has fourteen data types supported natively be ActiveRecord[15]. Ruby on Rails also has many active record validations supported natively. To fit the scope of the project not all data types and validations will be supported. The selection of data types and validations supported are as follows.

Data Types Supported

1. Integer
2. Float
3. String

Integer and Float Validations

1. Greater than
2. Greater than or Equal to
3. Equal to
4. Less than or Equal to
5. Less than
6. Other than
7. Divisible by
8. Blank
9. Inclusion
10. Exclusion

String Validations

1. Maximum Length
2. Minimum Length
3. Exact Length
4. Format

Table 1: Formal Specification Requirements

ID	Requirement	Priority
1	A user can create a Project	**M**
2	A user can edit a Project	**M**
3	A user can delete a Project, associated Tables are also deleted	**M**
4	A user can create a Table, only for a given Project	**M**
5	A user can edit a Table	**M**
6	A user can delete a Table, associated Fields are also deleted	**M**
7	A user can create a Field, only for a given Table	**M**
8	A user can edit a Field	**M**
9	A user can delete a Field, associated validations are also deleted	**M**
10	A user can assign a validation and value, only for a given Field	**M**
11	A value must be valid for a validation to be assigned	**M**
12	A user can edit a validation assignment and value	**M**
13	A user can delete a validation assignment and value	**M**
14	A user can view all Tables associated to a given Project	**D**
15	A user can view all Fields associated to a given Table	**D**
16	A user can view all validations and values associated to a given field	**D**

5. Blank
6. Inclusion
7. Exclusion

Table 2: RSpec Test Case Requirements

ID	Requirement
1	RSpec test case should be runnable
2	RSpec test case should only test one behaviour
3	RSpec test case must should test behaviour intended
4	RSpec descriptor must be human readable
5	When test case fails, its output must specify exact behaviour at fault
6	RSpec test case should have human readable syntax
7	Be able to generate an Integer that satisfies all validations and their values assigned to a
8	Be able to generate an Integer that does not satisfy a validation but satisfies all other val
9	Be able to generate a Float that satisfies all validations and their values assigned to a fiel
10	Be able to generate a Float that does not satisfy a validation but satisfies all other valida
11	Be able to generate a String that satisfies all validations and their values assigned to a fie
12	Be able to generate a String that does not satisfy a validation but satisfies all other valida

Table 3: RSpec Test Suite Requirements

ID	Requirement	Priority
1	RSpec test suite should be runnable	**M**
2	RSpec test suite should be contain all test cases for a table	**M**
3	RSpec test cases should be grouped via field	**D**
4	RSpec test cases should be in a logical order	**D**
5	RSpec test suite must be available for download	**M**

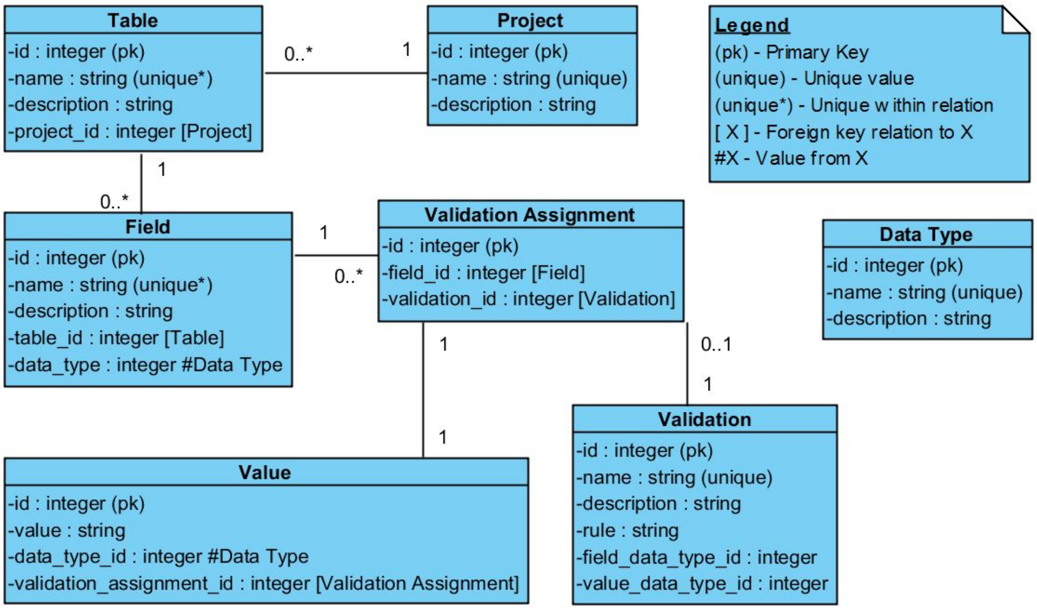

Figure 3: UML Database diagram

4 Chapter 4: Design

4.1 Database

The database was designed to accomodate the formal specification and extra information needed for RSpec test cases. Ruby on Rails has native support for relations via ActiveRecord, this provided flexibility in being able to have many relations. The design paradigm of sole responsiblity was carried forward and each table has one purpose.

Project

Project allows for a user to use the system for multiple projects, that is to organise groups of tables that are disconnected. The name field is unique to avoid confusion amongst projects. The description field is only for the user to add extra information if they desire and serves no functional purpose in the system. A project can have zero to many tables. When a project is deleted its associated tables should also be deleted.

Table

Table is the table in a user's database for which they are constructing a functional specification. A table can only be created with a valid project_id foreign key. The name is unique within the scope of the project, multiple tables can exist with the same name within the database but they must belong to seperate projects. The description field is only for the user to add extra information if they desire and serves no functional purpose in the system. A table belongs to a project and has zero to many fields. When a table is deleted its associated fields should also be deleted.

Field

Field is the field in a user's database for which they are constructing a functional specification. A field can only be created with a valid table_id foreign key. The name is unique within the scope of the table, multiple fields can exist with the same name within the database but they must belong to seperate tables. The description field is only for the user to add extra information if they desire and serves no functional purpose in the system. Data_type_id is the fields data_type and is limited to values in the Data Type table, however no relation is forced. A field belongs to a table and has zero to many Validations through Validation Assignments. When a table is deleted its associated validation assignments should also be deleted.

Validation

Validation is a validation the user can associate to a field. The user can not create, edit or destroy these and they are seeded in the database. The name is unique within the scope of its field_data_type. The description field is to aid the user in understanding what the validation is. The rule field is used within the system to generate values. Field_data_type_id and value_data_type_id limited to values in the Data Type table, however no relation is forced, and are used so only appropriate validations can be assigned to fields. A validation has zero to many fields through validation assignments.

Value

Value is the value of a validation that a user associates with a field, E.g false for blank or ten for minimum length. A value can only be created with a valid validation_assignment_id foreign key. The value is stored as a string and the data type of which the system should treat the value as is stored in data_type_id, which refers to the data type table, but no relation is enforced. A value belongs to a validation assignment.

Validation Assignment

Validation assignment associates a field with a validation and also the value for that validation. The assignment is created then a value is created belonging to the assignment. A validation assignment can only be created with both valid field_id and validation_id foreign keys. A validation assignment has one field, one validaiton and owns a value.

Data Type

Data type is used to provide consistency throughout the system by checking data types are equal between entities, checking values of are the correct type and also used in forms to reduce options available to the user. The user can not create, edit or destroy these and they are seeded in the database. The name field is unique to avoid confusion amongst data types and avoid possible duplication. The description field is to inform the user on the data type. Data type has no relations, but is used throughout the system in reference.

4.2 View and Flow

The main consideratons when designing the look and how the user navigates and uses the system were human-centered design principles with the goal of increasing effectiveness, efficency and satisfaction[16] .The principles taken into consideration and how they were applied are[17]:

1. **Simplifying the structure of tasks**

   An average user is able to remember five things at a time. Providing consistency within similar methods such as creation of entitys and also clearly displaying where in the system the user is reduces the strain on both long and short term memory. The creation, deletion and editing of entitys follows the familiar and consistent method of filling in a form and submitting it. At the top of each page it clearly displays how deep in the system and also the names of each level. This means the user will have to remember less when using the system.

2. **Exploiting the power of constraints**

   Reducing and not exposing a user to redundant or irrelevant information allows a user to use a system more efficently and with less effort, by not having to process such information. This is used by displaying only the related entities when viewing an entity, only the fields for a table are shown when viewing that table. When a user is creating an entity with a form options are reduced to those that are valid, only choose from Integer validations for a Integer field. Entitys that are dependent on a another, field is dependent on table, are only available to create when viewing the dependent upon entity.

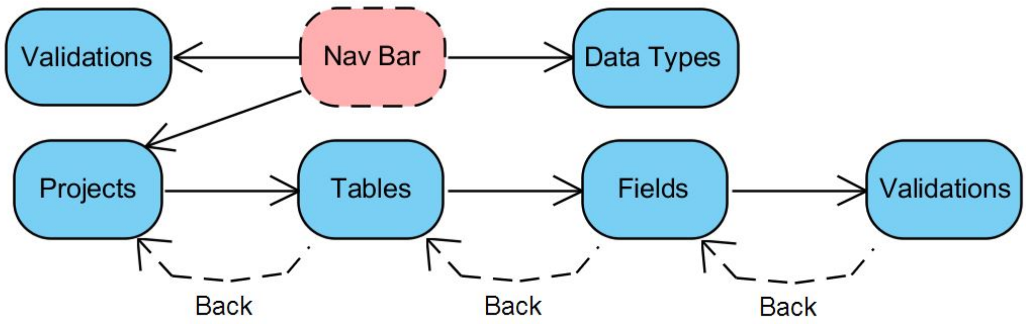

Figure 4: The navigational flow of the system

3. **Make things visible**

   Bridging the gulf of execution, buttons and links do what a user expects them to do. This principle followed by using standard naming conventions on standard objects such as buttons and links that a user is familiar with.

General

Consistency was at the the forefront of the visual and flow design. Each page that has a dependent entity can only be created via a button on the page of that entity6. When viewing an entity it will display all and only its dependent entities in a table.6 The dependent entites can be viewed, edited and deleted from this table via links in the row of that entity on the table7. Each page that is dependent upon on entity can access the dependent enity via the back button7, also applies naviagtionally to forms8. The header at the top of each page displays the depth the user is at by underlying the current level while also indicating the levels with the name of the entity at that level7.

Nav bar

The nav bar is not a page but the navigational bar displayed at the top of each page5. It links to the projects page, validations page and the data types page via the appropiatley name button.

FUNCTIONALITY OF PAGES ???

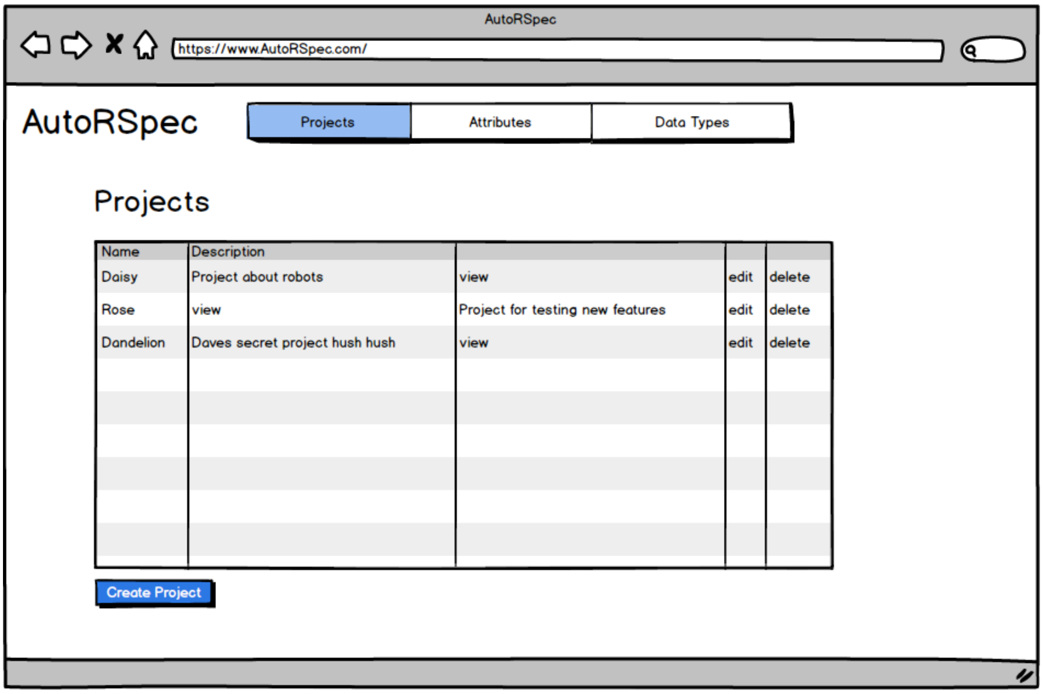

Figure 5: Mock up of projects page for the system, also home page.

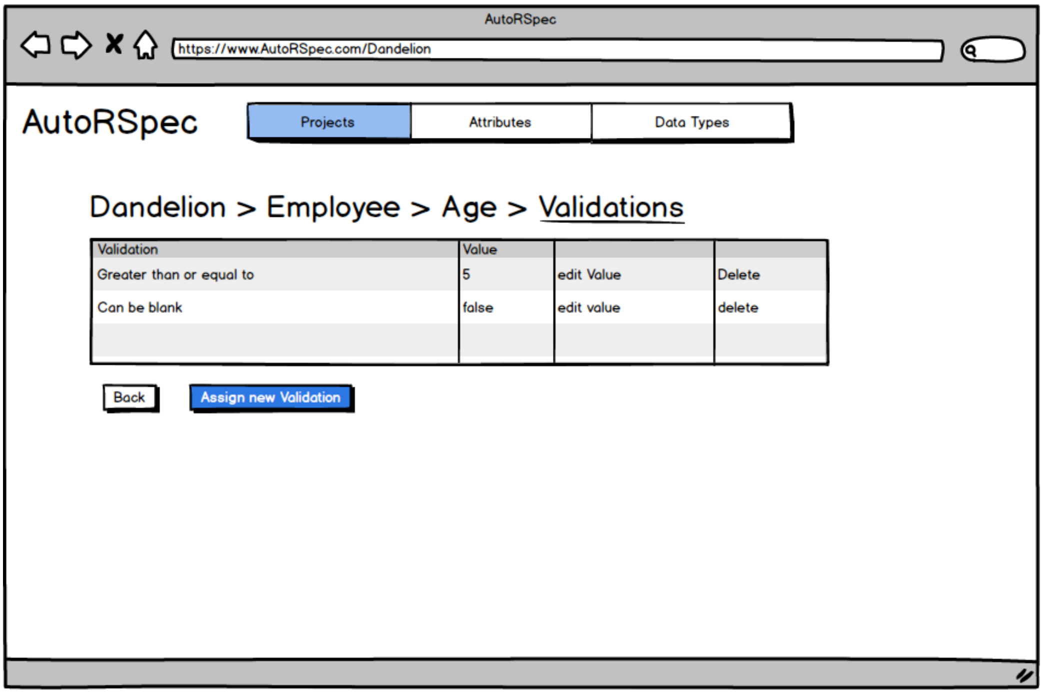

Figure 6: Mock up for a table in the system displaying its fields

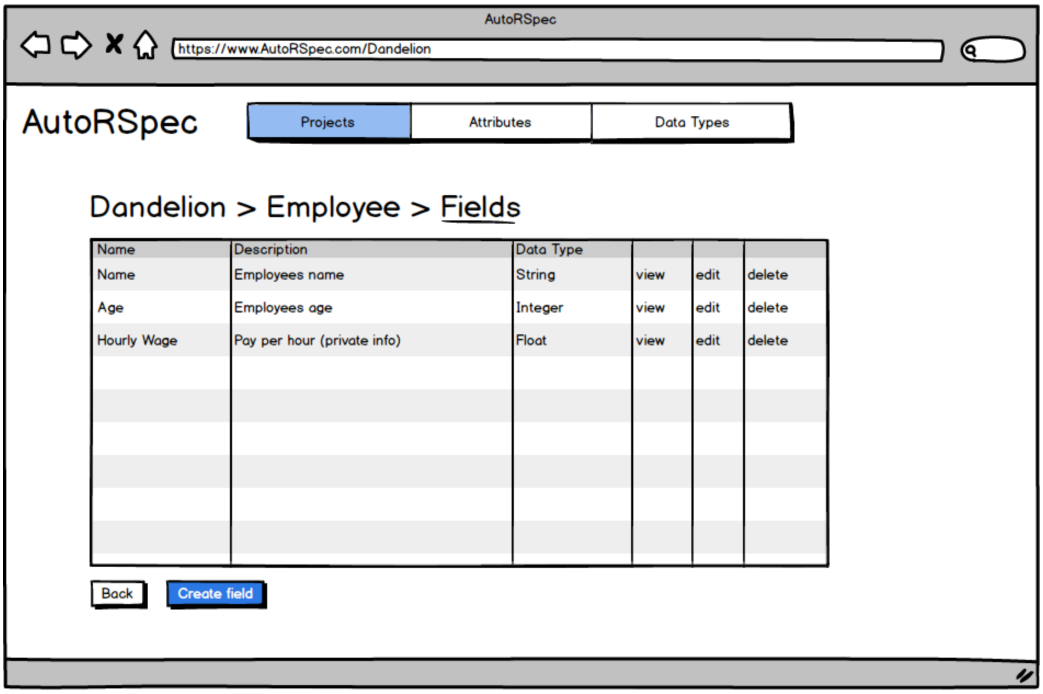

Figure 7: Mock up for a field in the system displaying its validation

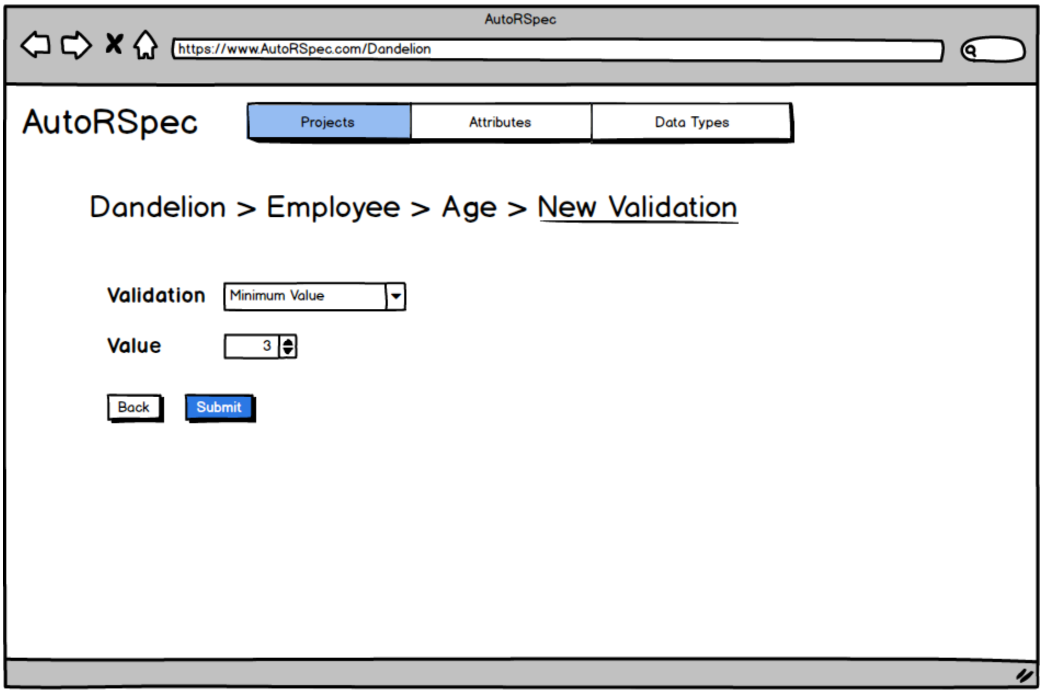

Figure 8: Mock up for a form in the system

4.3 Test Case Generation

4.3.1 Value Generation

Listing 1: Pseudo code for value generation label

```
generate_value( isolatedValidation, validations)
 for iterationCount in 1 to X
  randomValues = generate_random_values Y
   if isolatedValidation not nil
    randomValues = fail_validation(isolatedValidation, randomValues)
     if randomValues is empty
      next
   for validation in validations
    randomValues = pass_validation(validation, randomValues)
     if randomValues is empty
      next
   if randomValues is empty
     next
   else
    return random value from randomValues
 return ERROR NO VALUE GENERATED
```

The psuedo code in Listing 1 shows then general approach to generating a value for a test case. To increase the efficency and speed of potentailly returning a valid value by generating **X** values at a time for a potentail of **Y** times. Generating **X*Y** values initailly will take a greater deal of time and may be overkill. The correct balance will need to be found in implementation of the initail amount of values for each iteration and how many iterations to carry out.

The function will take two parameters, a validation to fail and a list of validations to pass. The validation to fail can be nil, in which case it will produce a value that passes all validations in validations parameter. The validations parameter can also be an empty list in which case it will return a value that fails the isolated validation. In the case of nil validation and and empty list of validations or no value can be generated it will return an error.

The flow of the function is to generate a random amount of values, then remove those values that pass the isolated validation, afterwards it will remove those that fail each validation in validations. After each time this array of random values is manipulated it will check if its empty, if so it will skip to the next iteration, this improves speed and efficency by eliminating unnesscary operations. When all validations have been processed it will return a random value from random values. The general flow and principles is the same for each data type that values will be generated.

4.3.2 RSpec Test Case

Listing 2: Pseudo code for value generation label

```
it "is (invalid with a value that is not VALIDATION VALVALUE" do
  TABLENAME = build(:TABLENAME, FIELDNAME: GENVALUE
  if TABLENAME.respond_to?(:valid?)
   expect(TABLENAME.not_to be_valid, lambda TABLENAME.errors.ful
 end
end
```

VALIDATION is the validation that is being tested on the field, and is described as its polar opposite

VALVALUE is the value of the validation that is being tested on the field.

TABLENAME is the name of the table the field belongs to

FIELDNAME is the name of the field under test

GENVALUE is the value that will test the validation

The RSpec test case template that will be populated with information from the functional specification and value generated. An example of a employees age field with validation of greater than value eight with a generated value of thirteen.

Listing 3: Pseudo code for value generation output label

```
it "is invalid with a value that is not greater than 8" do
  employee = build(:employee, age: 13)
  if employee.respond_to?(:valid?)
   expect(employee.to_not be_valid, lambda employee.errors.full_messag
 end
end
```

4.3.3 RSpec Test Suite

The test suite is responsible for the overall document structure that the user downloads. It will provide the preamble, then iterate through each field and finally end the document. It will provide all necessary logic to navigate test cases correctly and add the correct syntaxical links to the file.

Listing 4: Pseudo code for value generation label

```
1  require ”spec_helper”
2
3  desrcibe ”TABLENAMECAP” do
4   it ”has a valid factory” do
5    expect(build(:TABLENAME)).to be_valid
6   end
7   #for each field belonging to table |f|
8    describe ”f.NAME has property” do
9    # genval = generate_value(nil, f.validations)
10   # add to file test_case_pass_all(f,genval)
11   #for each validation belonging to f |v|
12    # genval = generate_value(v, rest of validations)
13    # add to file test_case(v,f,genval)
14   #end
15   end
16  #end
17 end
```

EXAMPLE ??

To make the pseudo code more clear lines starting with # are not printed to the document but code that is ran. TABLENAMECAP is the table name capitalised. Lines 1-4 are the preamble and set up the RSpec document and write a test to check the tables factory is valid. Then for each field it will add the correct syntax to the file for that field and follow it by adding a test case to the file that passes all validations. For each validation that field has been assigned it will iterate through isloting each field and adding the relevant test case to the file. Test suite is the top level function therefor before it calls to generate a test case, lines 8,11, it will call the function to generate a valid value for that test case, lines 7, 10. Test suite will finally end the document with the appropiate ”end” syntax that it opened in the preamble.

```
C:\Users\Dan\Desktop\Dissertation\AutoRspec>rails g scaffold field name:string description:string project_id:integer
      invoke  active_record
      create    db/migrate/20170501161227_create_fields.rb
      create    app/models/field.rb
      invoke    test_unit
      create      test/models/field_test.rb
      create      test/fixtures/fields.yml
      invoke  resource_route
       route    resources :fields
      invoke  scaffold_controller
      create    app/controllers/fields_controller.rb
      invoke    erb
      create      app/views/fields
      create      app/views/fields/index.html.erb
      create      app/views/fields/edit.html.erb
      create      app/views/fields/show.html.erb
      create      app/views/fields/new.html.erb
      create      app/views/fields/_form.html.erb
      invoke    test_unit
      create      test/controllers/fields_controller_test.rb
      invoke    helper
      create      app/helpers/fields_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/fields/index.json.jbuilder
      create      app/views/fields/show.json.jbuilder
      create      app/views/fields/_field.json.jbuilder
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/fields.coffee
      invoke    scss
      create      app/assets/stylesheets/fields.scss
      invoke  scss
   identical    app/assets/stylesheets/scaffolds.scss
```

Figure 9: Rails g scaffold command for field

5 Chapter 5: Implemention and Testing

5.1 Setup

Ruby version 2.1.7 and rails 4.2.5.2 were used to construct this project.

Github[18] was used throughout the project to provide version control and to prevent any catastrophes. The airbnb ruby style convention was used to provide clean looking and consistent code[19]. Rails new was used to instatiate the system. Then the generate scaffolfd command was used to construct the base for each table in the database schema outlined in chapter 4.1 9. This sets up the default mvc components for each, the exact files created can be seen in Figure 9.

Regular expressions are used frequently through the implementation process, to quickly check a regular expression was serving the function expected the Rubular[20] website was used to quickly verify them, this was not for testing but to provide further clarity on complex expressions. The gem Annotate[21] was used to annotate the models with database information such as field names to help efficency when implementing as the information is all in the same place.

```ruby
belongs_to :table
has_many :validation_assignments, dependent: :destroy, inverse_of: :field
has_many :validations, through: :validation_assignments
```

Figure 10: Rails native relation methods used in field.rb model

```ruby
validates :name,
  format: {with: /\A[a-z][a-z_]*[a-z]\Z/, message: "Must follow lower case underscore naming convention"},
  presence: true,
  uniqueness: {scope: :table_id, message:"Field already present in table"}

validates :table, presence: :true

validates :data_type_id, presence: :true
```

Figure 11: Rails native validation method used in field.rb models

5.2 Database

Rails natively supported active record assocations were used to establish the relations between the tables in the database[22]10. The dependent destroy option was used on relations where the deletion of the entity desired the deletion of entites that depended upon it outlined in the requirements, this will cause a cascade effect downwards through relations10. Validations were applied intially according to the requirements then also dynamically throughout the process as they came relevant through implementation, e.g a fields name should follow the lower case underscore format that is convention in naming table fields in Ruby on Rails applications**??**. These validations were for the functionallity of the test suite generation and related functions.

5.3 View and Flow

Bootstrap[23] was used for the css of the web pages. Bootstrap gives a very clean and presentable wesbite by adding default css to HTML elements. Further classes that were used were table, table-hover, button classes and navbar classes. Bootstrap is extremely efficent for the view aspect of implementation.

To get the flow outline in design the routes file had to be heavily configured. Rails reads the routes file from top to bottom, on the first path it hits that a match it will route to. Placing the custom routes above the nesting resources effectively overrides the defaults that rails will produce. The defaults needed to be overrode due to the nature of how when viewing an enity it displayed all of its dependent entities, aswell as being able to create a dependent entity.

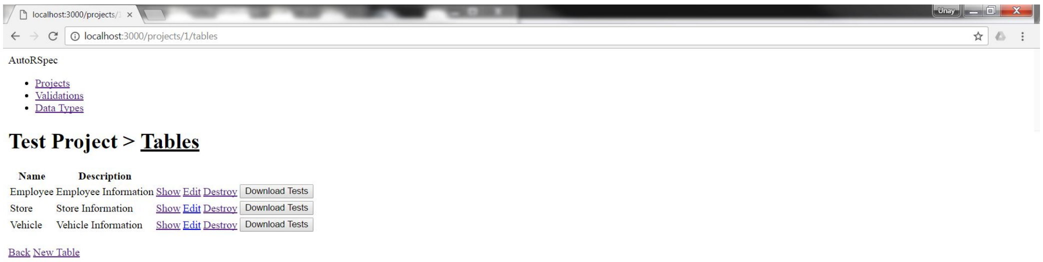

Figure 12: Tables page without bootstrap

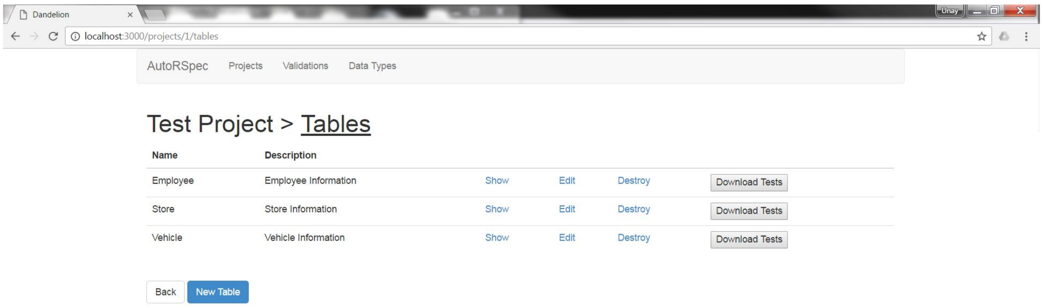

Figure 13: Tables page with bootstrap

When the routes file had been configured rails it created the paths that could be used throughout the project15. The path helpers are desired as they take variables and produce a link thefore aid in the scaling and readablity of the project whilst also providing a very clear and methodical link.[24]

5.4 Test Case Generation

5.4.1 Value Generation

The design for generating a value was modified slightly due to finer details but its overall flow and princple was kept. One oversight from the design was the exclusion validation, when an exculsion validation is to be isolated it is logical to just return a random value from the list of values excluded as

```
resources :validations
resources :data_types

#Button for generating tests
post "tables/download/:project_id/:id" => "tables#download"
#new
get 'projects/new' => 'projects#new'
get 'projects/:project_id/tables/new' => 'tables#new'
get 'projects/:project_id/tables/:table_id/fields/new' => 'fields#new'
get 'projects/:project_id/tables/:table_id/fields/:field_id/validation_assignments/new' => 'validation_assignments#new'
get 'projects/:project_id/tables/:table_id/fields/:field_id/validation_assignments/:validation_assignment_id/values/new' => 'values#

#show => index
get 'projects/:project_id' => 'tables#index'
get 'projects/:project_id/tables/:table_id' => 'fields#index'
get 'projects/:project_id/tables/:table_id/fields/:field_id' => 'validation_assignments#index'


resources :projects do
  resources :tables do
    resources :fields do
      resources :validation_assignments do
        resources :values
      end
    end
  end
end
```

Figure 14: Routes file for project

```
<%= link_to 'Back', project_table_fields_path(@project, @table, @field), class:"btn btn-default" %>
<%= link_to 'New Validation assignment', new_project_table_field_validation_assignment_path(
  @project, @table, @field), class: "btn btn-primary"  %>
```

Figure 15: Example of path helper

they will all fail the validation rather than try to randomly generate a value in the list.

String Generation

Random strings are generated using the 'regexp-examples'[25] gem that takes a regular expression and generates and array of strings that are valid to the expression.

Another gem 'to_regexp' gem[26] is used to

5.4.2 RSpec Test Case

5.4.3 RSpec Test Suite

6 Chapter 6: Results and Discussion

6.1 Evaluation

Most projects lit did are too complex and dont have much validation in terms of mine

AutoRSpec part of design process to be fully utilised, with the tests being built upon TDD.

Amahi, My software Hut, make dummies based upon those too complex with validations

1. Remove validations, one, few, all
2. Additional validations, supported, unsupported
3. Unsupported validations effect

1. time taken to produce formal specification
2. amount of test cases produced
3. readability of failing tests
4. readablity of test suite
5. buildability as a foundation

DOG FEED - biased and unscientifc, im used to system, my tests and my designed

time taken for me to write my tests

time taken for me to func spec

time saved

the amount of tests covered, any not covered, proportions

6.2 Future

6.2.1 Improvements to current system

1. User freindly - Conflicting rules - Error handling - Nav via heirachy line
2. border test cases
3. increase database fectchin efficeny

6.2.2 System additions

1. more validations

2. relations
3. more data types
4. generating customisabel/ all tables etc etc
5. could use knowledge to produce factory girl skeletons

6.2.3 Functional Specification

Import, Export, Use it

7 Chapter 7: Conclusions

References

[1] C. Barker, "The top 10 it disasters of all time," 2007.

[2] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[3] J. Gliek, "A bug and a crash," 1996.

[4] P. A. McQuaid, "Software disastersunderstanding the past, to improve the future," *Journal of Software: Evolution and Process*, vol. 24, no. 5, pp. 459–470, 2012.

[5] S. Nidhra and J. Dondeti, "Blackbox and whitebox testing techniques-a literature review," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.

[6] M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*. Wiley India Pvt. Limited, 2008.

[7] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[8] R. on Rails, "A guide to testing rails applications," 2017.

[9] Django, "Django faq," 2017.

[10] SHUUP, "25 of the most popular python and django websites," 2015.

[11] Alexa, "The top 500 websites," 2017.

[12] Coderfactory, "Top 15 sites built with ruby on rails," 2017.

[13] Infoq, "Facebook: Mvc does not scale, use flux instead," 2014.

[14] Ruby, "About ruby," 2016.

[15] R. on Rails, "Activerecord::connectionadapters::schemastatements," 2017.

[16] M. Maguire, "Methods to support human-centred design," *International journal of human-computer studies*, vol. 55, no. 4, pp. 587–634, 2001.

[17] C. Abras, D. Maloney-Krichmar, and J. Preece, "User-centered design," *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, vol. 37, no. 4, pp. 445–456, 2004.

[18] G. Hub, "Git hub," 2017.

[19] Airbnb, "Ruby style guide," 2017.

[20] Rubular, "Rubular a ruby regular expression editor," 2017.

[21] C. Tran, "Annotate models," 2017.

[22] R. on Rails, "http://guides.rubyonrails.org/association_basics.html," 2017.

[23] Bootstrap, "Bootstrap," 2017.

[24] R. on Rails, "Rails routing from the outside in," 2017.

[25] T. Lord, "regexp-examples," 2017.

[26] S. Abshere, "to_regexp," 2017.