

National College of Ireland
BSc in Computing
2015/2016



mtgCardWallet

Michele Gravina
X12106623

Michele.Gravina@student.ncirl.ie

Technical Report

Table of Contents

Executive Summary	6
Introduction	7
1.1 Background.....	7
1.2 Aims.....	7
1.2.1. Final Outcome	8
1.2.2. Methodology	8
1.2.3. Resources	8
1.3 Technologies	9
1.3.1. Android L	9
1.3.2. Android Studio	9
1.3.3. XML.....	9
1.3.4. PostgreSQL.....	9
1.3.5. SQLite	9
1.3.6. GitHub	9
1.4 Structure	10
System.....	10
2.1 Requirements	11
2.1.1. Functional requirements	11
2.2 Design and Architecture.....	17
2.2.1. System Architecture	17
2.2.2. Architecture Design	18
2.2.3. Use Case Diagram	19
2.3 Implementation	20
2.3.1. DataBaseHelper	20
2.3.2. CardSearch Symbols.....	20
2.3.3. Cookie and Session management.....	22
2.3.4. Authorization	23
2.3.5. Dynamic Header	23
2.4 Testing	23
2.4.1. Unit Testing	23
2.4.2. Test Driven Development (TDD)	24
2.5 Graphical User Interface (GUI) Layout.....	26
2.6 Evaluation	33
2.6.1. Human Evaluation	33
2.6.2. Machine Evaluation	33
Conclusions	33
Further development or research	33
Bibliography	34
Appendix.....	34
6.1 Project Proposal	34
Objectives	34
Background.....	35
Technical Approach	35
Special resources required	35
Project Plan.....	36

Technical Details.....	36
Evaluation	37
6.2 Requirement Specifications	38
Introduction	40
1.1 Purpose	40
1.2 Scope.....	40
1.3 Abbreviations and Definitions	41
1.4 Overview.....	43
The Overall Description.....	43
2.1 Product Perspective.....	43
2.2 User Classes and Characteristics	43
2.3 Constraints.....	44
2.4 Operating Environment	44
2.5 Assumptions and Dependencies	44
Requirements.....	45
3.1 Functional Requirements	45
3.2 External Interfaces.....	51
3.3 Nonfunctional Requirements.....	57
6.3 User Feedback Form	58
6.4 Monthly Journals.....	59
September 2015.....	59
October 2015.....	61
November 2015.....	62
December 2015.....	65
January 2016.....	69
February 2016	71
March 2016	71
April	72

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name:
Student ID:
Supervisor:

SECTION 2 **Confirmation of Authorship**

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarized overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: _____ Date: _____

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

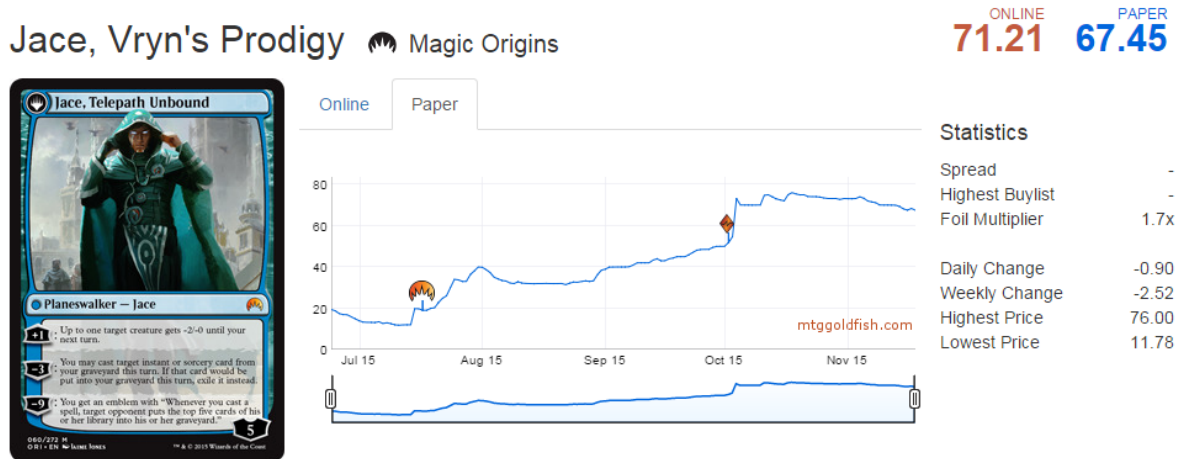
- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- the requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

Executive Summary

“mtgCardWallet” is a supporting Android Application that is built upon the success of the trading card game Magic: The Gathering with over 15440 individual cards (Wizards, 2015) that fluctuate in price, similar to stocks as seen in the image below.



(mtggoldfish.com, 2015)

“mtgCardWallet” will allow a User to organize and maintain his *Magic: The Gathering* trading card collection more efficiently without the need of an active Internet connection and without the hassle of a registration.

Once a user's opens the App, he will be directly able to add cards to his collection. The application will provide an updated Database that includes all data related to each individual Magic the Gathering card and each Set to enable a smooth creation process of the collection.

The User provides the name of the card and mtgCardWallet will highlight all available version. The user selects the condition of the card, how many copies and any additional data such as promotional, signed by the artist or altered. Once this process is completed for the whole collection “mtgCardWallet” will provide the following main services.

- Details on Personal Collection
- Collection Value and individual cards
- Creation of Tournament Decks
- Card Wish List
- Life Counter

In addition to these main features, several additional features will be provided in a later stage. Such as:

- Daily Update of Pricing Information
- Easy Sale / Purchase of Cards on magiccardmarket.eu
- MTG Big Data Analytics on Collection
- Suggestion for additions to collection
- Feedback on played Tournaments

- Tournament Scheduler
- Recent Tournament Deck lists
- Available cards for recent Tournament Decks
- Deck suggestions based on availability of cards

The Application will be funded by participating in the affiliate Program for magiccardmarket.eu and based on donations. There are options to enter additional markets and provide same services for other successful trading card games such as Pokémon, Star Wars, NetRunners etc. There are currently no plans to implement advertisement. Possible competitors are: mtgprice.com and deckbox.org but none of those are providing the same service as “mtgCardWallet”.

Introduction

With the recent 20 Year anniversary and increasing success year after year, it is becoming essential to utilize technology and find a viable alternative to the trading card management.

The aim of this project was to create an application for organizing and maintaining your Magic: The Gathering collection, the main criteria being user friendliness in terms of both simplicity and design.

This report presents the designs for the mtgCardWallet application and it's components such as: Database, User Interface and development environment. It is aimed at players and collectors, independent of the size of their collection. mtgCardWallet supports large and small collections alike.

1.1 Background

I'm an active part of the Irish Magic: The Gathering Community. This includes traveling internationally to events and expanding a collection of thousands of Paper *Magic: The Gathering* Cards to play with and collect. Cards can maintain a value of several hundred Dollars, fluctuating just like the stock market. It is important to keep an eye on your collection to know when to buy and sell certain Cards to maximize your efficiency.

The game is played usually in an assortment of 60 cards, that's called a Deck. People like me can have multiple Decks as subsets of their collection of cards. This makes it difficult to keep track of where cards are currently being used, how many copies are owned of a particular card and also what version of the card is available (language, promotional etc.). I often found myself in a situation that I purchased cards, just to find out I already owned the card, resulting in a waste of Money. mtgCardWallet is an application that allows me to keep track of my Magic: The Gathering Collection and will support me organizing it.

1.2 Aims

The Goals of this project is to create a functional Android application that allows to create, update and delete cards in a user's Magic: The Gathering collection that is stored in a local Database on his device. Once the collection has been entered in mtgCardWallet, the User will have access to all of the Information of his collection without the need of an Internet Connection.

1.2.1. Final Outcome

Project is considered complete when the mobile Application Interface is working and a user can edit maintain and edit a collection using his Android device.

1.2.2. Methodology

Agile methodology has been used to develop mtgCardWallet. The Project has been broken down into several Sprints of features that have been refined prior and planned accordingly for. After each sprint the success of the feature has been reviewed in a refinement session and action plans created based on the outcome of the session.



(strategicstaff.com, 2016)

1.2.3. Resources

Users will need access to Android L compatible mobile devices such as the Samsung Galaxy S7. It will require around 100MB of space on the phone and relied on Gatherer, a tool provided by Wizards of the Coast to access all data about specific cards.

1.3 Technologies

The project is implemented in a number of parts; the bulk of the Work is done using Android Studio and the Android SDK, the main Mobile User Interface is designed for Android L and the Navigation Drawer in the main Activity as highlighted by the material design specifications. (google.com, 2016), is done using XML. A splash Image to show the mtgCardWallet Logo upon startup of the Application is also implemented using XML.

Floating Action Buttons are used to provide an intuitive user Interface experience and to utilize features of the latest Android L technology.

The project Database was initially designed in PostgreSQL but then moved to SQLite for compatibility reasons with Android. A Database Helper class is implement using Java to provide the application access to the Database and Version Control is implemented using GitHub.

1.3.1. Android L

As Google's Material design has been respected, I wanted to utilize its latest features and decided to develop and optimize the Application to Android's latest release. Android L.

1.3.2. Android Studio

Android Studio has been used to do all development work for the mobile Application.

1.3.3. XML

XML is a vital part of Android Development. User Interface, Resources such as Strings, Colors and even shapes are all implemented in Android using XML.

1.3.4. PostgreSQL

PostgreSQL is a sophisticated free a MySQL DB that has been chosen as default Database for its ease of integration with Heroku and Rails.

1.3.5. SQLite

As the project has been moved from Rails to Android, the Database had to be converted to SQL as Android provides a framework to easily access a SQLite Database from your application.

1.3.6. GitHub

GitHub has been chosen over other version control software as previous projects have been successfully run without complications and the low ramp up time and ease of use were additional factors that contributed to the decision of GitHub.

1.4 Structure

This document has been structured such that by reading the Introduction section, the reader will be provided with a general overview and concepts of the Application. The following section will detail to the reader about the background of the Application, what the main aims of the Application are and continues then with an overview of technologies used in its creation.

The third section will provide details on functional and non-functional requirements, describing them and how they will be addressed individually. It will also highlight concerns and issues associated with each of the requirements. Dependencies to other requirements and risks will be addressed by this section, too.

In the next section, the Report will detail the Architecture and Design of the Application. How the components of the Application have been tested and how the User Interface has been designed?

Finally, it will cover how the Application the Application has been tested by members of the Magic: The Gathering tutorial and what feedback has been received by their evaluation. It will move on the list the conclusions made after completing the project and what further development is planned to improve the application and end up at the bibliography and used resources that contributed to the completion of the application.

The final chapter will contain the appendices, original submitted project proposal. The requirement specification document, monthly journals and any additional appendices

System

The "mtgCardWallet" System consist of two major parts. The Database that contains all details about the trading Cards, Sets, Collection, Decks and Wish list. The Mobile Application used as the primary interface for Users to interact with the System with its simplistic User Interface. As more cards are added to Magic: The Gathering the application provides easy to use and upgradable process to keep the Card Database up to date. A relational Database is used to centralize all the Data required by different parts of the System, Interfaces have to be established to Read, Write, Delete and Edit Data within the Database directly by Interacting with the User Interface. Security mechanics have been put in place to ensure those interfaces only serve its purpose

The Diagrams in this section highlights how the individual parts of the System interact with each other. Since this System follows a data-centric approach to its implementation the Collection, Users, Decks and Cards have to be centralized and permanently stored within the System. For that, a database will be used. Both the mobile application and web portal will communicate with the

2.1 Requirements

2.1.1. Functional requirements

2.1.1.1 *Functional Requirement 1*

Magic: The Gathering card and set database.

2.1.1.2 *Description & Priority*

This is the heart of the Application that includes data about every *Magic: The Gathering* card existing. A user can only include cards into his collection that are included in the card Database.

2.1.1.2 *Activation*

When a User opens the Application, a check is performed if an updated card database exists, if that is false it will build a new SQL Database that contains all details and data about cards and sets. It then sets up an empty collection, wish list and deck's table. The Application allows for maintenance of the collection by the user.

2.1.1.3 *Technical Issues*

As a lot of data is accumulated, the System requires enough memory and hard disk space on the smartphone to perform the action. If the smartphone has not enough space, the Database cannot be build and the Application not used.

2.1.1.4 *Risks*

As the data is stored locally on the smartphone of the user, if the data is corrupted the collection cannot be recovered.

2.1.1.5 *Dependencies*

For users to create collections the Card and Set data is required. This user requirement is essential to mtgCardWallet.

2.1.1.6 *Code Segment*

This code segment details how the MTG_sets and MTG_cards table is created in the Database and what data is associated to each set and card.

Check if DB exists:

```
//create Database if data does not already exist  
myDbHelper = new DataBaseHelper(this);
```

```

try {
    myDbHelper.createDataBase();
} catch (IOException ioe) {
    throw new Error("Unable to create database");
}
try {
    myDbHelper.openDataBase();
} catch (SQLException sqle) {
    throw sqle;
}

```

MTG_Sets table:

```

CREATE TABLE MTG_sets(MTG_set_name text,MTG_set_code text PRIMARY
KEY,MTG_set_code_magiccards text,MTG_set_date text,MTG_set_is_promo
text,MTG_set_boosterpack_nM text,MTG_set_boosterpack_nR text,MTG_set_boosterpack_nU
text,MTG_set_boosterpack_nC text,MTG_set_boosterpack_nE text,MTG_set_boosterpack_pM
text,MTG_set_boosterpack_pR text,MTG_set_boosterpack_typeExtra1
text,MTG_set_boosterpack_typeExtra2 text,MTG_set_boosterpack_listExtra1
text,MTG_set_boosterpack_listExtra2 text,MTG_set_boosterpack_has_foil text,MTG_set_boosterpack_pF
text);

```

MTG_cards table:

```

CREATE TABLE MTG_cards(MTG_card_id text PRIMARY KEY,MTG_card_name text,MTG_card_set
text,MTG_card_type text,MTG_card_rarity text,MTG_card_manacost text,MTG_card_converted_manacost
text,MTG_card_power text,MTG_card_toughness text,MTG_card_loyalty text,MTG_card_ability
text,MTG_card_flavor text,MTG_card_variation text,MTG_card_artist text,MTG_card_number
text,MTG_card_rating text,MTG_card_ruling text,MTG_card_color text,MTG_card_generated_mana
text,MTG_card_pricing_low text,MTG_card_pricing_mid text,MTG_card_pricing_high
text,MTG_card_back_id text,MTG_card_watermark text,MTG_card_print_number
text,MTG_card_is_original text,MTG_card_name_CN text,MTG_card_name_TW text,MTG_card_name_FR
text,MTG_card_name_DE text,MTG_card_name_IT text,MTG_card_name_JP text,MTG_card_name_PT
text,MTG_card_name_RU text,MTG_card_name_ES text,MTG_card_name_KO text,MTG_card_type_CN
text,MTG_card_type_TW text,MTG_card_type_FR text,MTG_card_type_DE text,MTG_card_type_IT
text,MTG_card_type_JP text,MTG_card_type_PT text,MTG_card_type_RU text,MTG_card_type_ES
text,MTG_card_type_KO text,MTG_card_ability_CN text,MTG_card_ability_TW text,MTG_card_ability_FR
text,MTG_card_ability_DE text,MTG_card_ability_IT text,MTG_card_ability_JP text,MTG_card_ability_PT
text,MTG_card_ability_RU text,MTG_card_ability_ES text,MTG_card_ability_KO text,MTG_card_flavor_CN
text,MTG_card_flavor_TW text,MTG_card_flavor_FR text,MTG_card_flavor_DE text,MTG_card_flavor_IT
text,MTG_card_flavor_JP text,MTG_card_flavor_PT text,MTG_card_flavor_RU text,MTG_card_flavor_ES
text,MTG_card_flavor_KO text,MTG_card_legality_Block text,MTG_card_legality_Standard
text,MTG_card_legality_Modern text,MTG_card_legality_Legacy text,MTG_card_legality_Vintage
text,MTG_card_legality_Highlander text,MTG_card_legality_French_Commander
text,MTG_card_legality_Tiny_Leaders_Commander text,MTG_card_legality_Commander
text,MTG_card_legality_Peasant text,MTG_card_legality_Pauper text,FOREIGN KEY (MTG_card_set)
REFERENCES MTG_sets(MTG_set_code));

```

2.1.3.1 *Functional Requirement 2*

Magic:The Gathering collection maintenance

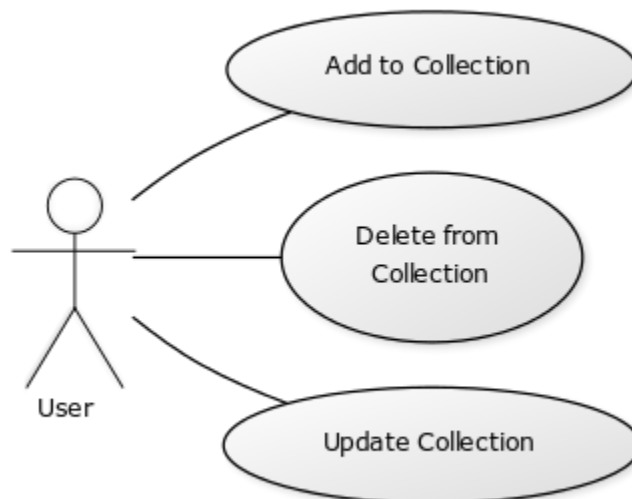
2.1.3.2 *Description & Priority*

This application is used to quickly add, remove and update cards in the collection by navigating the user interface. It is essential that the user can perform any action (create, read, update and delete) on his collection as deemed necessary without causing problems to his collection.

As it is the core functionality of mtgCardWallet the priority is as critical as FR1.

2.1.3.3 *Activation*

When a User opens the Application and the database has been successfully created, the user is directed to the collection screen that provided UI elements to add cards quickly into his collection.



2.1.3.4 *Technical Issues*

Cards can have identical properties and duplication cannot be avoided.

2.1.3.5 *Risks*

Navigating through the User Interface requires a touchscreen and a running smartphone. Without them, the collection cannot be accessed.

2.1.3.6 *Dependencies*

For users to create collections FR01 and FR03 is required.

2.1.3.7 Code Segment

This code segment shows how data is being added to the user's collection.

```
public void addToCollection(String cardID, int amount, String language, String
condition, int foil, int signed, int altered){

    SQLiteDatabase db = getReadableDatabase();
    if (db == null) {
    }

    db.execSQL("INSERT INTO collections VALUES (NULL,'" +cardID+"', '"+language+"',
'"+condition+"', "+foil+", "+signed+", "+altered+", "+amount+");");
}
```

Delete card from collection:

```
public void deleteCardFromCollection(int id){

    SQLiteDatabase db = getReadableDatabase();
    if (db == null) {
    }

    db.execSQL("DELETE FROM collections WHERE _id = "+id+"; ");
}
```

Update card in collection:

```
public void updateCollection(String cardID, int amount, String language, String
condition, int foil, int signed, int altered, int id){

    SQLiteDatabase db = getReadableDatabase();
    if (db == null) {
    }

    db.execSQL("UPDATE collections SET MTG_card_id = '"+cardID+"', altered =
"+altered+", signed = "+signed+", foil = "+foil+", card_language = '"+language+"',
condition='"+condition+"', quantity = "+amount+" WHERE _id = "+id+"; ");
}
```

2.1.3.8 Functional Requirement 3

Magic: The Gathering card search

2.1.3.9 Description & Priority

The user has to find the exact card to add into his possession without spelling errors. Auto populated Text utilizes the User Interface so the user selects instead of providing a full card name. This is another core functionality of the Application.

2.1.3.10 Activation

When a User adds cards to his collection, the system needs to know what exact card to add so the user specifies the name of the card and the system provides an autocomplete textbox for the user to select the correct card name.

2.1.3.11 Technical Issues

Identical cards can be in many sets with the same properties, however these are still different cards and the user has to specify the exact card that will be added to the collection. This is ensured by accepting the foreign key constrains and pairing each card with a card_set attribute which results in a unique search result.

2.1.3.12 Risks

The performance of the search is slowing down the bigger the card database gets. Indexing is important to optimize the speed in which the results can be generated.

2.1.3.13 Dependencies

For users to search for cards, the FR01 is required.

2.1.3.14 Code Segment

This code segment shows how the search for cards works.

```
//method responsible to return autocomplete text
public ArrayList<String> getSearchResult(String name) {
    SQLiteDatabase db = getReadableDatabase();
    if (db == null) {
        return null;
    }

    searchResults = new ArrayList<>();
    //apply escape for string in case of special characters
    String qname= name.replaceAll("'", "'");

    cr = db.rawQuery("SELECT DISTINCT MTG_card_name FROM mtg_cards WHERE
MTG_card_name LIKE '%" +qname+"%' OR MTG_card_name LIKE '"+qname+" %" OR
MTG_card_name LIKE '%" +qname+" ' OR MTG_card_name LIKE '"+qname+"' ", null);

    if (cr.getCount() == 0) {
        searchResults.add("No Card Selected");
    }else {
        cr.moveToFirst();
    }
}
```

```

        searchResults.add(cr.getString(cr.getColumnIndex("MTG_card_name")));

        while (cr.moveToNext()) {
            //searchResults.add(cr.getString(cr.getColumnIndex("MTG_card_set"))+"
| " +cr.getString(cr.getColumnIndex("MTG card name")));
            searchResults.add(cr.getString(cr.getColumnIndex("MTG_card_name")));
        }
    }
    return searchResults;
}

```

Once the search identified results, it will fill an ArrayAdapter with the results and presents it to the user to choose from.

```

@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {

    array = myDbHelper.getSearchResult(cardSearchItems.getText().toString());

    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getContext(),
        android.R.layout.select_dialog_item, array);
    cardSearchItems.setAdapter(adapter);
}

```

Update card in collection:

```

public void updateCollection(String cardID, int amount, String language, String
condition, int foil, int signed, int altered,int id){

    SQLiteDatabase db = getReadableDatabase();
    if (db == null) {
    }

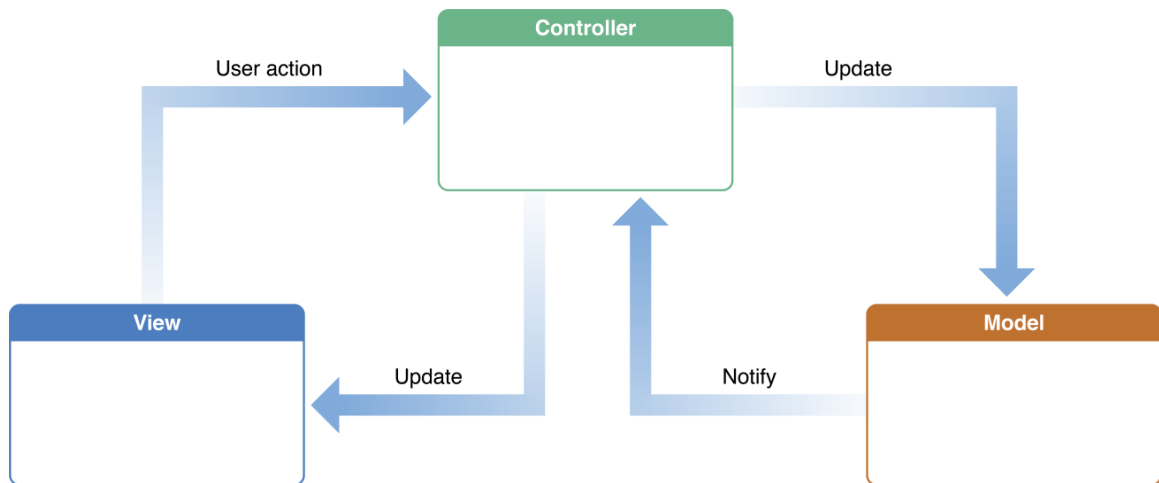
    db.execSQL("UPDATE collections SET MTG_card_id = '"+cardID+"', altered =
"+altered+", signed = "+signed+", foil = "+foil+", card_language = '"+language+"',
condition='"+condition+"', quantity = "+amount+" WHERE _id = "+id+"; ");
}

```


2.2 Design and Architecture

2.2.1. System Architecture

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries.

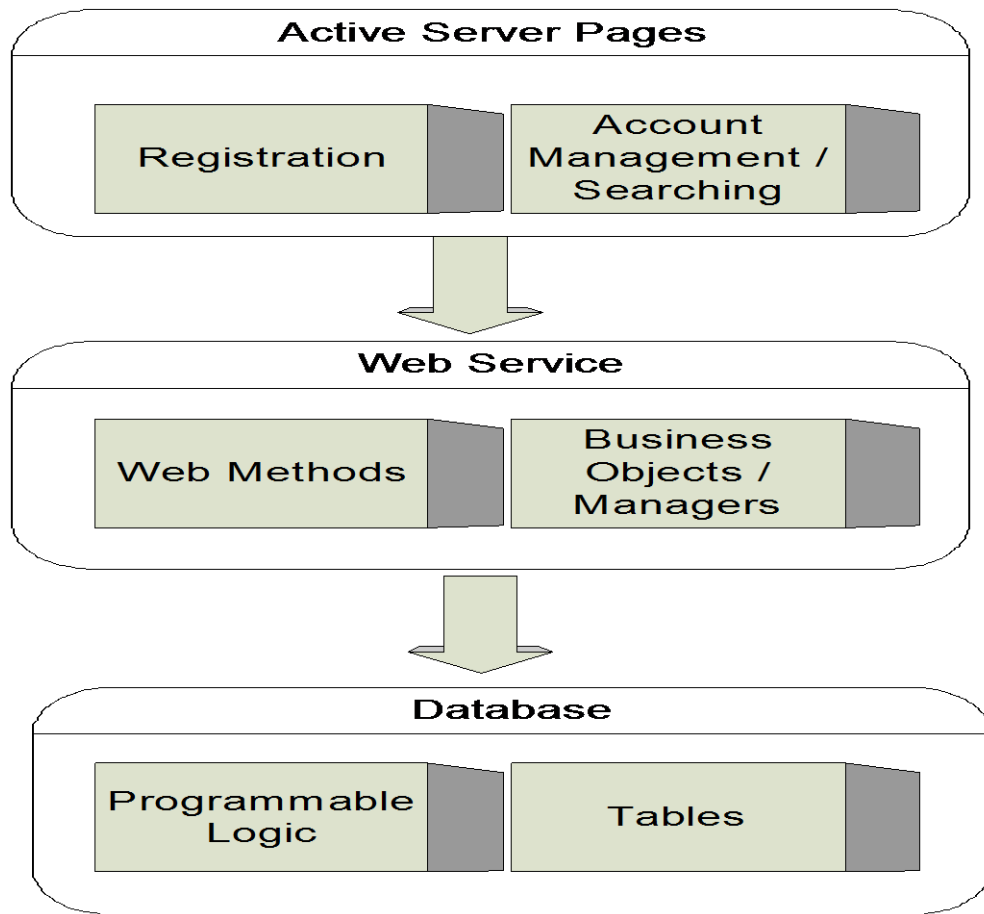


(Apple, 2015)

- View: How the System will end up looking on the phone. It includes all TextView, Buttons, Images etc. The controller implements the view to display the model.
- Model: The model is where all computation takes place.
- Controller: The controller takes the result from model and transforms it into the View.

Model-View-Controller (MVC)´s main task is to take all the objects from memory and place them in each of the above areas.

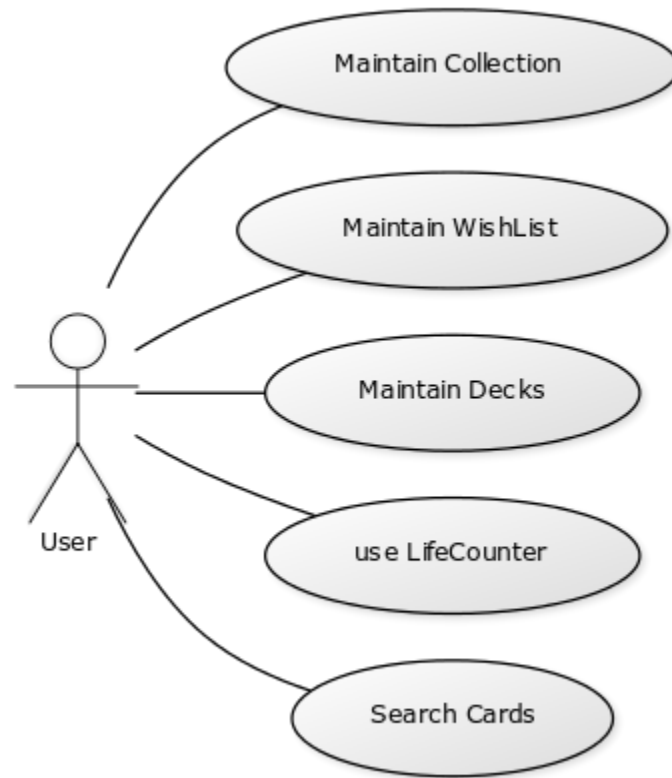
2.2.2. Architecture Design



(www.se.rit.edu, 2015)

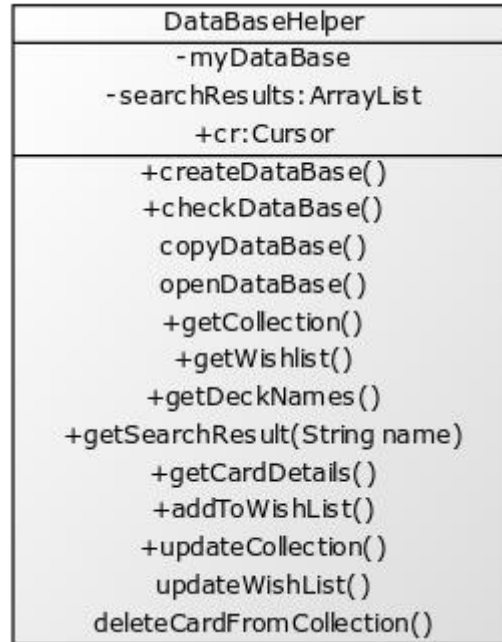
This layer contains all of the pages that the user sees, and some of the basic logic to present them and validate input. The second layer is the Web Service layer, this layer provides all Business Logic functions and connects the Presentation Layer and the Data Layer. The third Layer is the Data Layer, this layer consists of the Database, and utilizes stored procedures to keep the logic of maintaining the database closer to the data.

2.2.3. Use Case Diagram



2.3 Implementation

2.3.1. DataBaseHelper



This class provides the Interface and connection from the Application to the Database.

2.3.2. CardSearch Symbols

```
private static final Map<Pattern, Integer> emoticons = new HashMap<Pattern, Integer> ();
```

```
static {  
    addPattern(emoticons, "{C}", R.drawable.ic_colorless);  
    addPattern(emoticons, "{W}", R.drawable.ic_icon_white);  
    addPattern(emoticons, "{U}", R.drawable.ic_icon_blue);  
    addPattern(emoticons, "{B}", R.drawable.ic_icon_black);  
    addPattern(emoticons, "{R}", R.drawable.ic_icon_red);  
    addPattern(emoticons, "{G}", R.drawable.ic_icon_green);  
    addPattern(emoticons, "{0}", R.drawable.ic_icon_0);  
    addPattern(emoticons, "{1}", R.drawable.ic_icon_1);  
    addPattern(emoticons, "{2}", R.drawable.ic_icon_2);  
    addPattern(emoticons, "{3}", R.drawable.ic_icon_3);  
    addPattern(emoticons, "{4}", R.drawable.ic_icon_4);  
    addPattern(emoticons, "{5}", R.drawable.ic_icon_5);  
    addPattern(emoticons, "{6}", R.drawable.ic_icon_6);  
    addPattern(emoticons, "{7}", R.drawable.ic_icon_7);  
    addPattern(emoticons, "{8}", R.drawable.ic_icon_8);  
    addPattern(emoticons, "{9}", R.drawable.ic_icon_9);  
    addPattern(emoticons, "{10}", R.drawable.ic_icon_10);  
    addPattern(emoticons, "{11}", R.drawable.ic_icon_11);  
    addPattern(emoticons, "{12}", R.drawable.ic_icon_12);  
}
```

```

addPattern(emojicons, "{13}", R.drawable.ic_icon_13);
addPattern(emojicons, "{14}", R.drawable.ic_icon_14);
addPattern(emojicons, "{15}", R.drawable.ic_icon_15);
addPattern(emojicons, "{16}", R.drawable.ic_icon_16);
addPattern(emojicons, "{2W}", R.drawable.ic_icon_hybrid_2w);
addPattern(emojicons, "{2U}", R.drawable.ic_icon_hybrid_2u);
addPattern(emojicons, "{2B}", R.drawable.ic_icon_hybrid_2bl);
addPattern(emojicons, "{2R}", R.drawable.ic_icon_hybrid_2r);
addPattern(emojicons, "{2G}", R.drawable.ic_icon_hybrid_2g);
addPattern(emojicons, "{BG}", R.drawable.ic_icon_hybrid_bg);
addPattern(emojicons, "{BR}", R.drawable.ic_icon_hybrid_br);
addPattern(emojicons, "{GU}", R.drawable.ic_icon_hybrid_gb);
addPattern(emojicons, "{RG}", R.drawable.ic_icon_hybrid_rg);
addPattern(emojicons, "{RW}", R.drawable.ic_icon_hybrid_rw);
addPattern(emojicons, "{UR}", R.drawable.ic_icon_hybrid_ur);
addPattern(emojicons, "{WB}", R.drawable.ic_icon_hybrid_wb);
addPattern(emojicons, "{WU}", R.drawable.ic_icon_hybrid_wu);
addPattern(emojicons, "{GW}", R.drawable.ic_icon_hybrid_gw);
addPattern(emojicons, "{UB}", R.drawable.ic_icon_hybrid_ub);
addPattern(emojicons, "{PW}", R.drawable.ic_icon_phyrexia_w);
addPattern(emojicons, "{PU}", R.drawable.ic_icon_phyrexia_u);
addPattern(emojicons, "{PB}", R.drawable.ic_icon_phyrexia_b);
addPattern(emojicons, "{PR}", R.drawable.ic_icon_phyrexia_r);
addPattern(emojicons, "{PG}", R.drawable.ic_icon_phyrexia_g);
addPattern(emojicons, "{T}", R.drawable.ic_icon_tap);
addPattern(emojicons, "{S}", R.drawable.ic_icon_snow);
addPattern(emojicons, "{X}", R.drawable.ic_icon_x);
addPattern(emojicons, "{Q}", R.drawable.ic_icon_untap);

// ...
}

```

This code snippet above is to create a Hash map that will be used to replace certain character sequences with Magic the Gathering Icons.

```

private static void addPattern(Map<Pattern, Integer> map, String smile,
                               int resource) {
    map.put(Pattern.compile(Pattern.quote(smile)), resource);
}

```

This code add's the mao from pattern to resource into the map.

```

public static boolean addMTGIcon(Context context, Spannable spannable) {
    boolean hasChanges = false;
    for (Map.Entry<Pattern, Integer> entry : emojicons.entrySet()) {
        Matcher matcher = entry.getKey().matcher(spannable);
        while (matcher.find()) {
            boolean set = true;
            for (ImageSpan span : spannable.getSpans(matcher.start(),
                matcher.end(), ImageSpan.class))
                if (spannable.getSpanStart(span) >= matcher.start()
                    && spannable.getSpanEnd(span) <= matcher.end())
                    spannable.removeSpan(span);
            else {
                set = false;
                break;
            }
        }
    }
}

```

```

        if (set) {
            hasChanges = true;
            spannable.setSpan(new ImageSpan(context, entry.getValue()),
                matcher.start(), matcher.end(),
                Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
        }
    }
    return hasChanges;
}

public static Spannable getMTGIconText(Context context, CharSequence text) {
    Spannable spannable = spannableFactory.newSpannable(text);
    addMTGIcon(context, spannable);
    return spannable;
}

```

When the above methods are called, a String is passed into the method and compared against the map. It will replace all character sequences and replace it with the correct Icon. The end result is the below TextView.

Spiritmonger



Creature — Beast

AP

Whenever Spiritmonger deals damage to a creature, put a +1/+1 counter on Spiritmonger.



: Regenerate Spiritmonger.



: Spiritmonger becomes the color of your choice until end of turn.

*Rails Implementations

2.3.3. Cookie and Session management

```

# Remembers a user in the session.
def remember(user)
  user.remember

```

```
cookies.permanent.signed[:user_id] = user.id
cookies.permanent[:remember_token] = user.remember_token
end
```

2.3.4. Authorization

```
before_action :logged_in_user
before_action :correct_user
```

2.3.5. Dynamic Header

```
<header class="navbar navbar-fixed-top">
  <div class="container">
    <%= link_to "mtgCardWallet", root_path, id: "logo" %>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><%= link_to "Home", root_path %></li>
          <% if !logged_in? %>
        <li><%= link_to "Demo", demo_path %></li>
          <li><%= link_to "Log in", login_path %></li>
          <% else %>
        <li><%= link_to "My Collection", mycollection_path(current_user) %></li>
        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown">
            Account <b class="caret"></b>
          </a>
          <ul class="dropdown-menu">
            <li><%= link_to "Profile", current_user %></li>
            <li><%= link_to "My Collection", mycollection_path(current_user) %></li>
            <li><%= link_to "Account Settings", edit_user_path(current_user) %></li>
            <li class="divider"></li>
            <li>
              <%= link_to "Log out", logout_path, method: "delete" %>
            </li>
          </ul>
        </li>
        <% end %>
      </ul>
    </nav>
  </div>
</header>
```

2.4 Testing

2.4.1. Unit Testing

Whenever a feature was finished in development, I carried out unit testing to cross reference each Requirement with its final implementation. This involved running the application on my Android device and monitoring the 'All Output' in the Console to check for any flags the TDD test cases raise.

2.4.2. Test Driven Development (TDD)

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase

  def setup
    @user = User.new(name: "Example User", email: "user@example.com",
                     password: "foobar", password_confirmation: "foobar")
  end

  test "is user valid? " do
    assert @user.valid?
  end

  test "is name included?" do
    @user.name = "a" * 51
    @user.name = " "
    assert_not @user.valid?
  end

  test "is email included?" do
    @user.email = "a" * 244 + "@example.com"
    @user.email = " "
    assert_not @user.valid?
  end

  test "email validation valid?" do
    valid_addresses = %w[user@example.com USER@foo.COM A_US-ER@foo.bar.org
                        first.last@foo.jp alice+bob@baz.cn]
    valid_addresses.each do |valid_address|
      @user.email = valid_address
      assert @user.valid?, "#{valid_address.inspect} should be valid"
    end
  end

  test "reject invalid entry in email field" do
    invalid_addresses = %w[user@example.com user_at_foo.org user.name@example.
                           foo@bar_baz.com foo@bar+baz.com]
    invalid_addresses.each do |invalid_address|
      @user.email = invalid_address
      assert_not @user.valid?, "#{invalid_address.inspect} should be invalid"
    end
  end
end
```



```
test "check if address in unique" do
  duplicate_user = @user.dup
  duplicate_user.email = @user.email.upcase
  @user.save
  assert_not duplicate_user.valid?
end

test "transform email to lowercase" do
  mixed_case_email = "Foo@ExAMPlE.CoM"
  @user.email = mixed_case_email
  @user.save
  assert_equal mixed_case_email.downcase, @user.reload.email
end

test "password included?" do
  @user.password = @user.password_confirmation = " " * 6
  assert_not @user.valid?
end

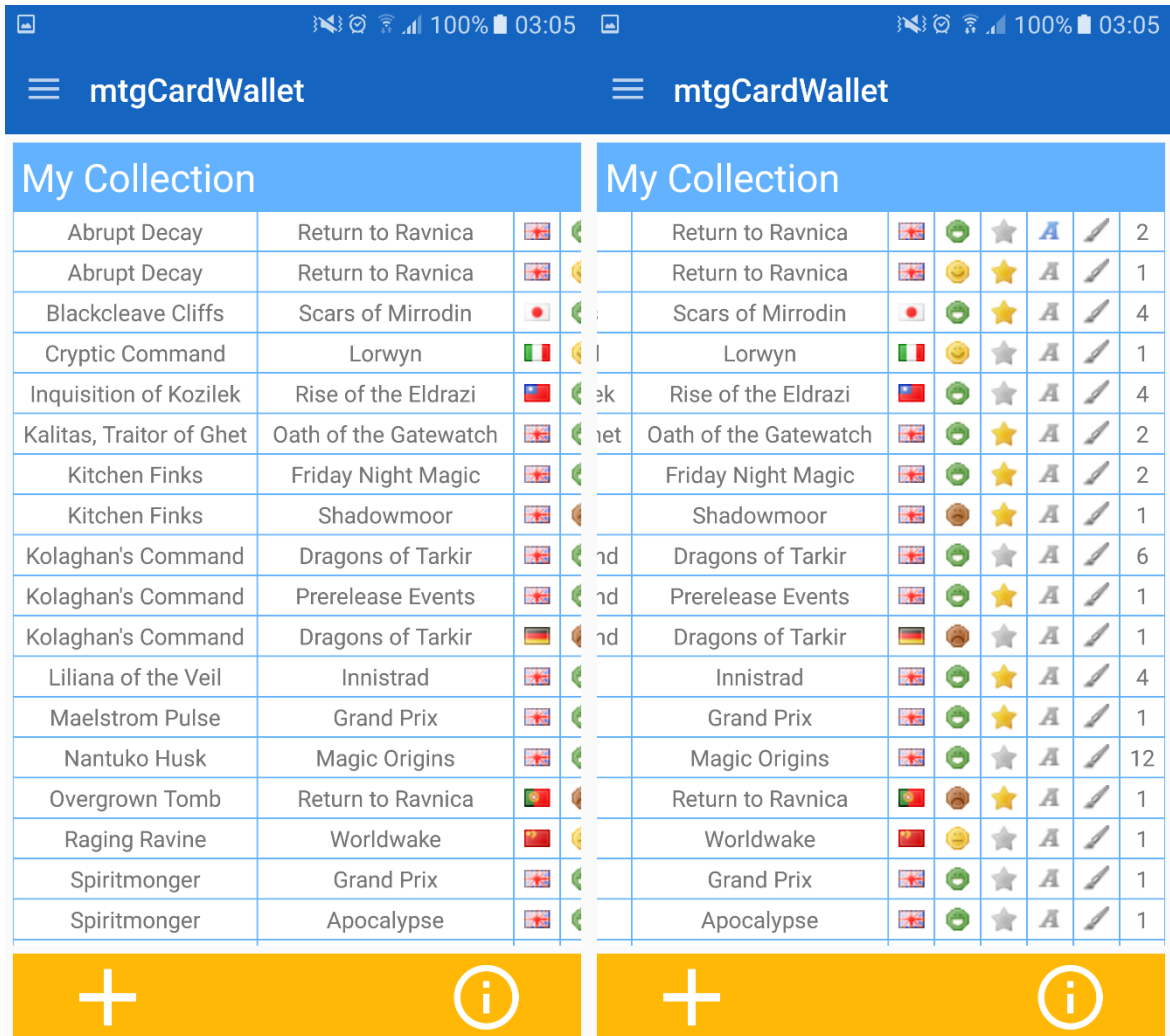
test "password minimum length" do
  @user.password = @user.password_confirmation = "a" * 5
  assert_not @user.valid?
end

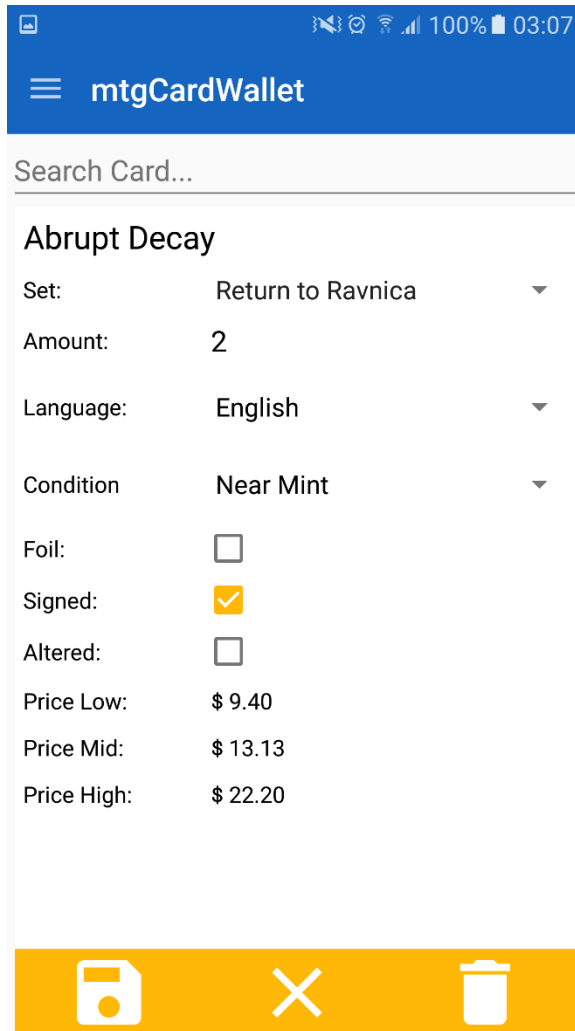
test "is user authenticated? should return false with nil digest" do
  assert_not @user.authenticated?("")
end

end
```

2.5 Graphical User Interface (GUI) Layout

Opening the “mtgCardWallet” App directs the user to the My collections page. This is a ScrollView and the User can scroll left/right and up/down to browse through his collection.





Clicking on an Individual Card opens the Edit menu to allow a user to update his collection, the User can also save changes directly. The cross in the middle provides the users with a reset function to set all the changes to its origin.

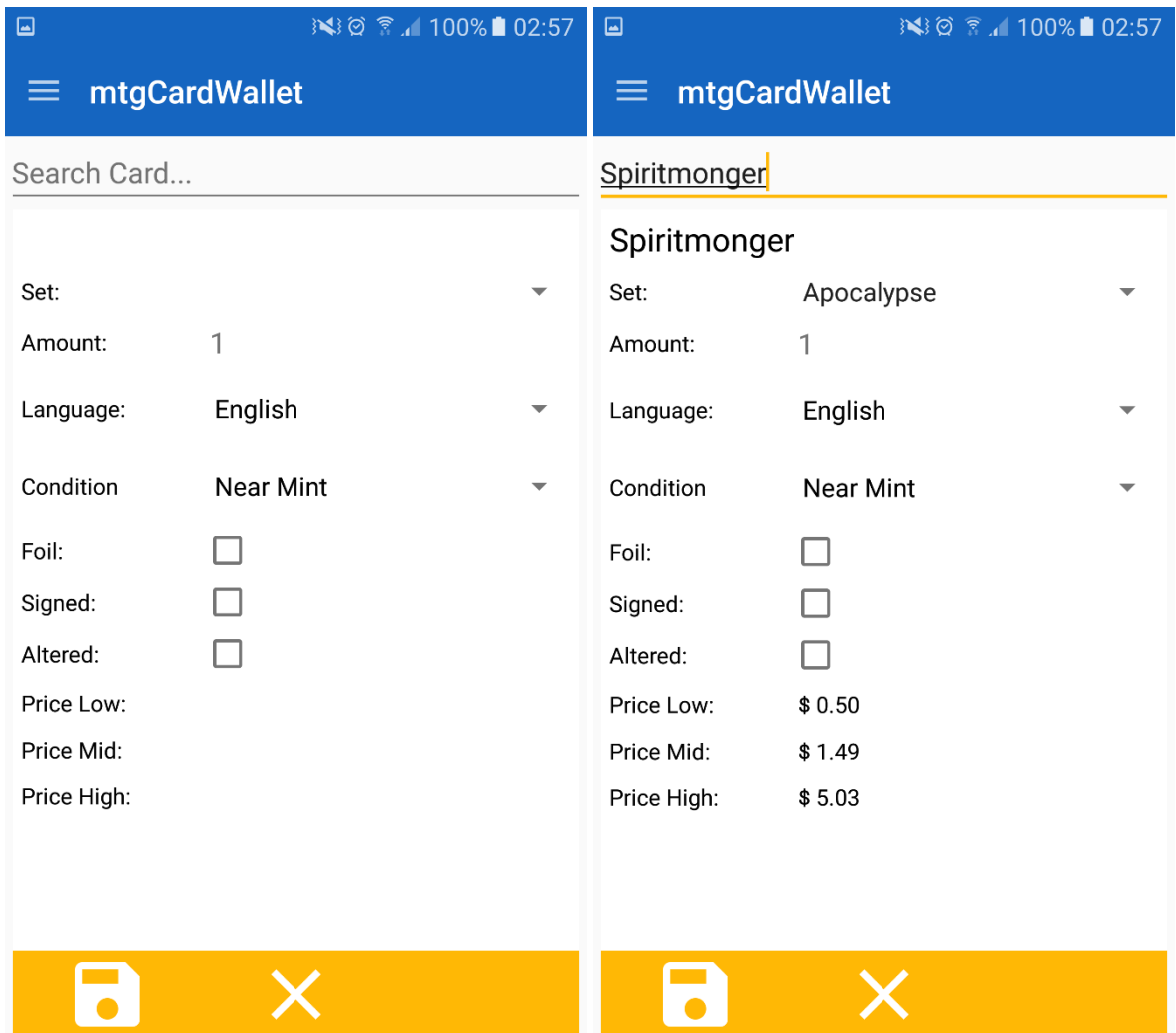
The icon in the bottom right corner opens the collections details menu. This menu shows useful information and a summary of the collection inspected.

The screenshot shows the 'mtgCardWallet' app interface. At the top, there is a blue header with a hamburger menu icon and the text 'mtgCardWallet'. Below this, there is a section titled 'Most Valuable Cards' with a table listing five cards and their values. The table has three columns: Card Name, Set Code, and Value. The cards listed are Tarmogoyf (FUT, \$189.05), Dark Confidant (JCG, \$129.32), Liliana of the Veil (ISD, \$110.00), Twilight Mire (EVE, \$39.68), and Thoughtseize (LRW, \$36.70). Below the table is a section titled 'My Collection Summary' with three rows of statistics: Total Cards: 65, Total low: \$1377.81, Total Value: \$2519.83, and Total high: \$5159.03. At the bottom of the screen, there is a large orange button with a white plus sign.

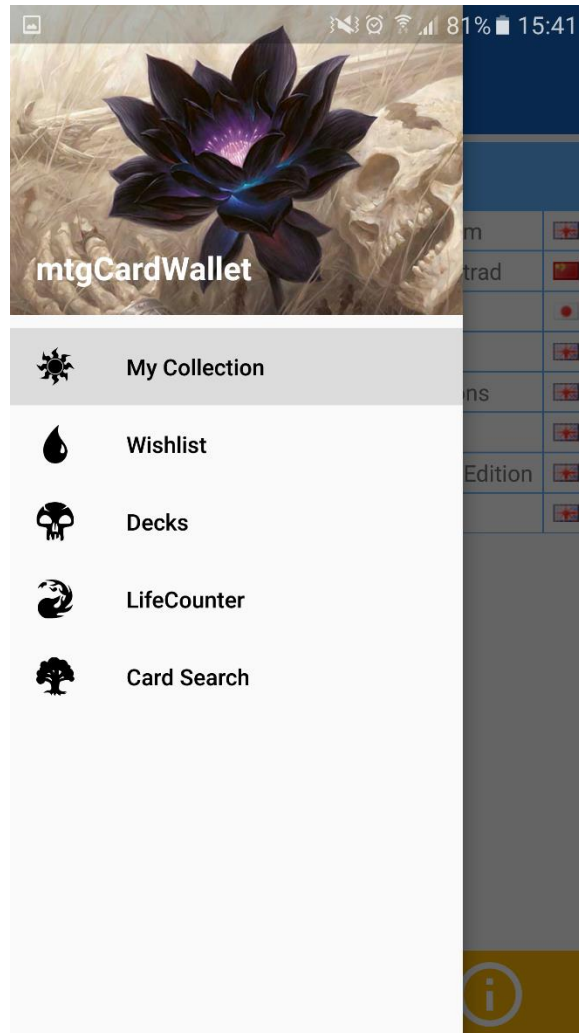
Most Valuable Cards		
Tarmogoyf	FUT	\$ 189.05
Dark Confidant	JCG	\$ 129.32
Liliana of the Veil	ISD	\$ 110.00
Twilight Mire	EVE	\$ 39.68
Thoughtseize	LRW	\$ 36.70

My Collection Summary		
Total Cards:	65	Total low: \$ 1377.81
		Total Value: \$ 2519.83
		Total high: \$ 5159.03

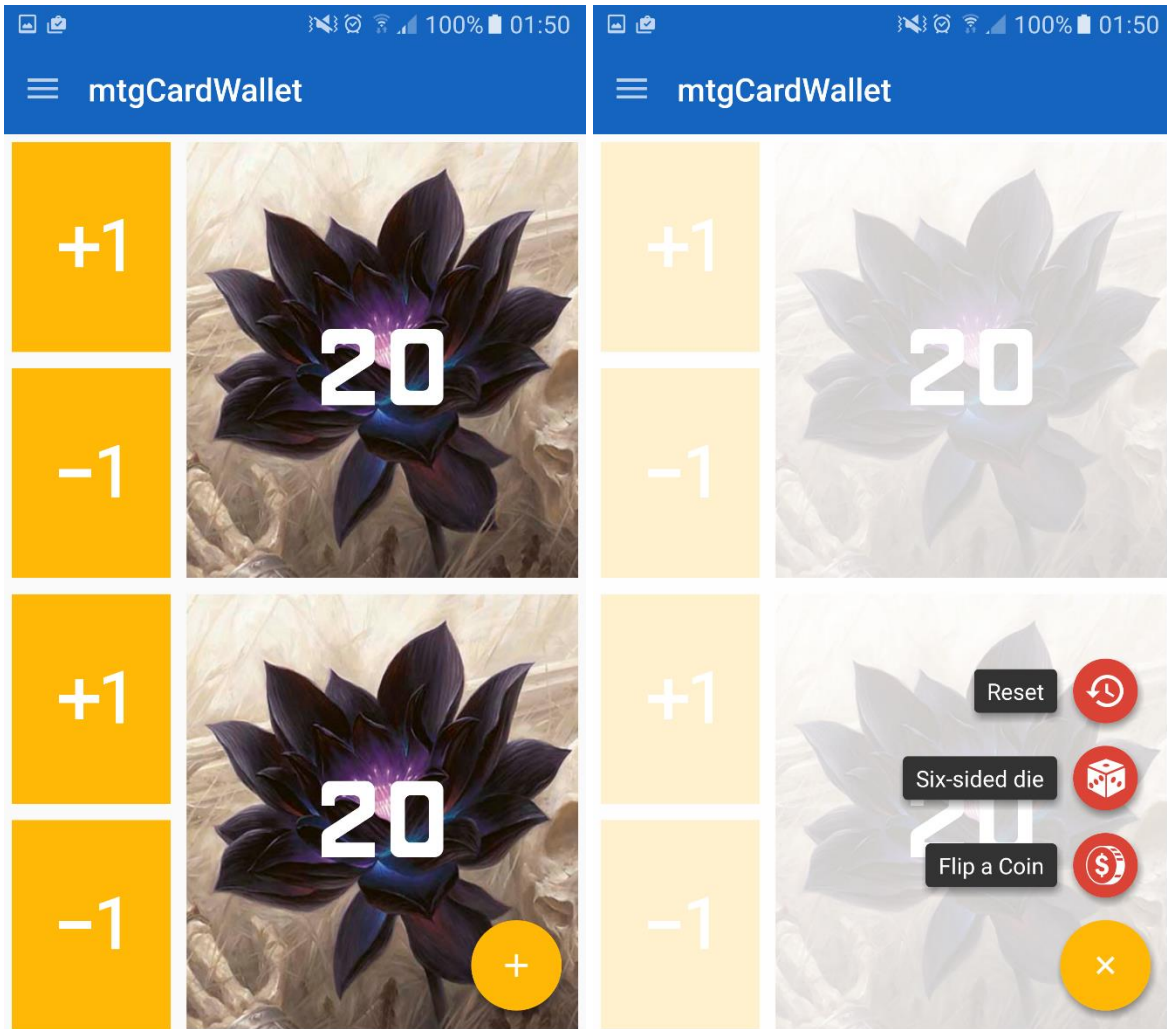
As with in the other menu, the user can add a card directly from here into his collection. A press on the bottom left icon opens the card search menu.



Searching a card populates the information about the card and populates the price information.





Pressing the drawer menu on the top left corner opens the Navigation menu. From here the user has always access to all features of the Application.




The Life Counter provides additional features to the user.


mtgCardWallet

Spiritmonger


Spiritmonger 3  
Creature — Beast AP

Whenever Spiritmonger deals damage to a creature, put a +1/+1 counter on Spiritmonger.

: Regenerate Spiritmonger.

: Spiritmonger becomes the color of your choice until end of turn.

Glen Angus 6 / 6
121



The card search utilizes the Card Database and allows the user to look up every available card and gain insightful Information including its rules text.

2.6 Evaluation

2.6.1. Human Evaluation

mtgCardWallet was evaluated using the free Service of Google Forms which is attached in the Appendix. 37 people participated. 11 Female and 26 males. The summary of the results obtained is:

- Design 6.3/10
- User Interface: 6.1/10
- Performance: 4.9/10
- Reliability: 6.7/10

- 7 People liked the User Interface the most.
- 11 People liked the Card Search.
- 5 People liked the Life Counter.
- 3 People liked the Performance.
- 11 People liked the Ease of Use.

12/37 Would recommend mtgCardWallet to a friend.

2.6.2. Machine Evaluation

```
2.8 Evaluation
2.8.1 All Test case results:
33/33: [=====] 100% Time: 00:00:00, Time: 00:00:00

Finished in 0.55242s
33 tests, 88 assertions, 0 failures, 0 errors, 0 skips
```

Conclusions

It was a good decision to move the development of mtgCardWallet to Android. A mobile application is more suitable for the needs of mtgCardWallet. I was able to utilize my Android Development experience. I'm happy with the look and feel of the Application and will continue working on it to add additional features that can contribute to a successful launch of the app.

Further development or research

The potential of this application is great. A successful implementation of Magic: The Gathering could lead to many more trading card games to follow. Pokemon, Yu-Gi-Oh! And Netrunner to just name a few. Each individual game would lead to additional Revenue sources by taking part in affiliate programs.

The advantages for mtgCardWallet are clear. A Service like this does currently not exist. There are similar functionalities available on Sites like mtggoldfish, mtgprice, mtgdecks etc but none of them utilize the features of a smartphone.

Bibliography

- Apple. (2015, 10 21). *developer.apple.com*. Retrieved 01 2016, from Cocoa Core Competencies: <https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>
- google.com. (2016, 04). Material Design - Patterns - Navigation drawer.
- mtggoldfish.com. (2015, 11 18). *mtggoldfish.com*. Retrieved from mtggoldfish.com: <http://www.mtggoldfish.com/price/Magic+Origins/Jace+Vryns+Prodigy#paper>
- strategicstaff.com. (2016). Agile Development.
- Wizards. (2015, 11 18). *Gatherer*. Retrieved from gatherer.wizards.com: [http://gatherer.wizards.com/Pages/Search/Default.aspx?name=+\[%22%22\]](http://gatherer.wizards.com/Pages/Search/Default.aspx?name=+[%22%22])

Appendix

6.1 Project Proposal

Objectives

The Goals of this project is to create a functional Web Application that allows a User to sign up for mtgCardWallet.

A signed up User can go to his collection and start entering Items into a list. After submitting this list, they are stored into the Database and his collection is created. He can adapt his Collection and apply different modifications to it, such as what Decks are currently using this card, how many, which versions and additional comments.

Final Outcome – Project will be considered as complete when the web and mobile version is working. With a usable interface, the Magic: The Gathering card Database is implemented and if users can add and edit their collection.

Methodology - I will use an agile methodology within the confines of the project milestones laid out by the project brief.

Resources – Will need access to computer, mobile devices such as phones and tablets. Access to MS Visio for development and Gatherer Extractor to create the Card Database. I will also need to attend lectures and research Ruby to complete the Project.

Monitor – Monitoring of the project will held by a Project Plan, I will assess regularly through at least once a month to highlight the month's progress and the next week's goals in an agile development and project methodology.

Background

I'm very actively playing the trading Card Game, Magic: The Gathering. This includes traveling internationally to events and expanding a collection of thousands of Paper Magic The Gathering Cards to play with and collect. Cards can maintain a value of several hundred Dollars, fluctuating just like the stock market. It is important to keep an eye on your collection to know when to buy and sell certain Cards.

The game is played usually in an assortment of 60 cards, that's called a Deck. People like me can have up to 10 decks in addition to their collection of cards. This makes it difficult to keep track of where the cards are being used at the moment, how many cards are owned of a particular one and also what version of the Card is available (language, promotional, foil etc). I found myself several times in a situation that I purchased cards, just to find out I already had them in one of the other Decks. mtgCardWallet is an application that allows me to keep track of my Magic: The Gathering Collection and help me organize it.

Technical Approach

The project will be implemented in a number of parts, there will be a web front end which will be on both a browser and a mobile app using bootstrap. This will have a database to store the content. I intend to use Ruby as a means for the web front end to talk to the database.

Implementations:

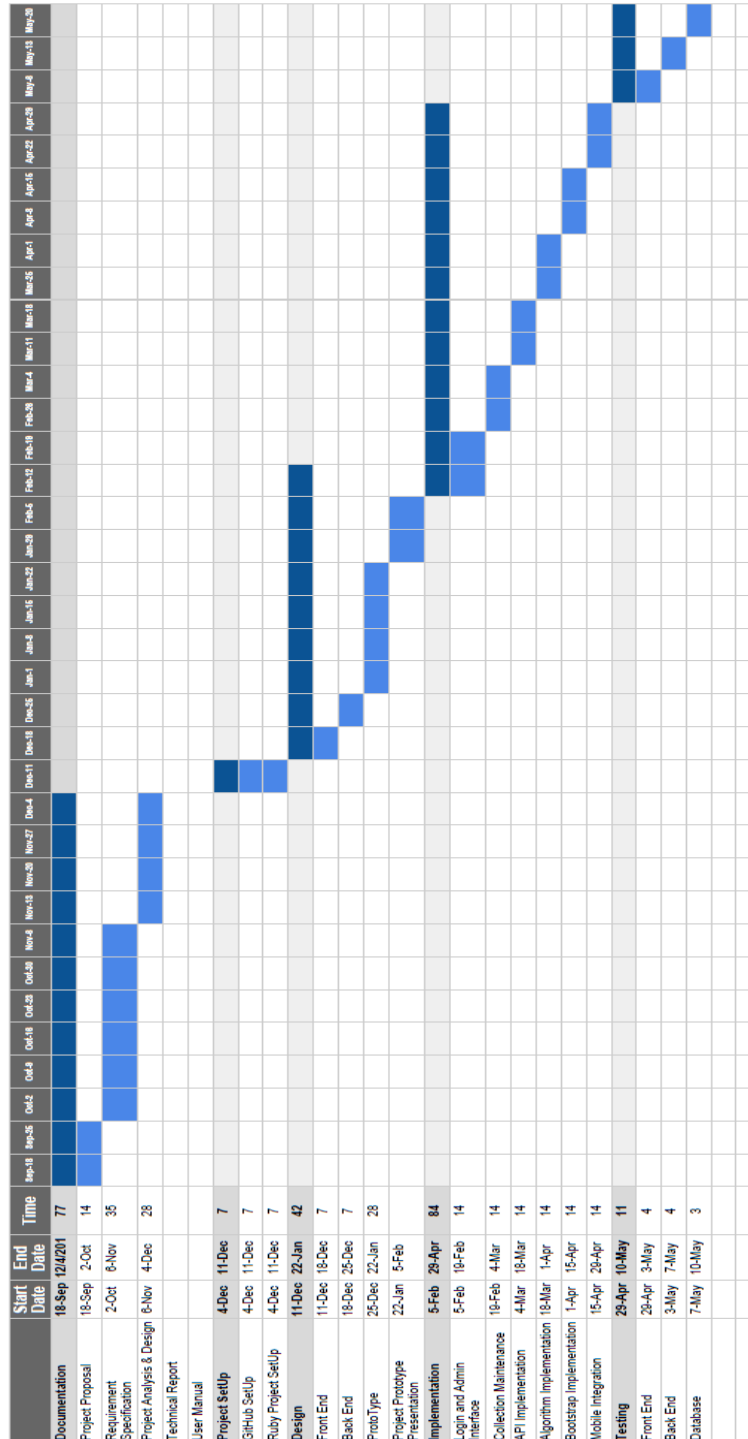
- SQL / Postgres Database
- Rails 4.2
- Frontend Bootstrap
- API (mtgcardmarket.eu)
- Web Services

Special resources required

Technical resources are an IDE and relevant web development libraries including twitter bootstrap and a SQL database, gems etc.

The largest resources will be time and human, I need to learn a lot and quickly. Research needs to be done to support the Project in all of its functionalities.

Project Plan



Technical Details

The project will be implemented in a number of parts, there will be a web front

end which will be on both a browser and a mobile app using bootstrap. This will have a database to store the content. I intend to use Visual Studio and C# as a means for the web front end to talk to the database.

Implementations:

- SQL / Postgres Database
- Backend Functionality
- Frontend Bootstrap
- Mobile Front End

Evaluation

In phase 1 every functionality will have a step by step Test Case prepared and I will go through all the steps and document my findings and improve the code directly or document and outline my refining phase and work in progress. If I'm happy with a functionality I will move it to phase 2.

In phase 2 my initial findings and documentation is passed on to a fellow magic the gathering player to re-evaluate and document his findings. I will adapt and make changes based on the results in phase 2.

Once the functionality passed phase 2 will move into the beta phase and is available publicly to test. I will use the magic the gathering community (reddit/mtg) to provide feedback and maintain a Bugzilla Account to keep track of bugs.

Methods will be imbedded to test the responds time of the Web Service including 10, 100, 10000 etc entries.

6.2 Requirement Specifications

Requirement Specifications

mtgCardWallet

Your trading card collection in your pocket.

Michele Gravina - x12106623

Table of Contents

1. Introduction	40
1.1 Purpose	40
1.2 Scope	40
1.3 Abbreviations and Definitions	41
1.4 Overview	43
2. The Overall Description	43
2.1 Product Perspective	43
2.2 User Classes and Characteristics	43
2.3 Constraints	44
2.4 Operating Environment	44
2.5 Assumptions and Dependencies	44
3. Requirements	45
3.1 Functional Requirements	45
3.2 External Interfaces	51
3.3 Nonfunctional Requirements	57

Introduction

This section provides clarification on purpose and scope for the web application “mtgCardWallet” and additionally provides an Overview of the application, common abbreviations and definitions.

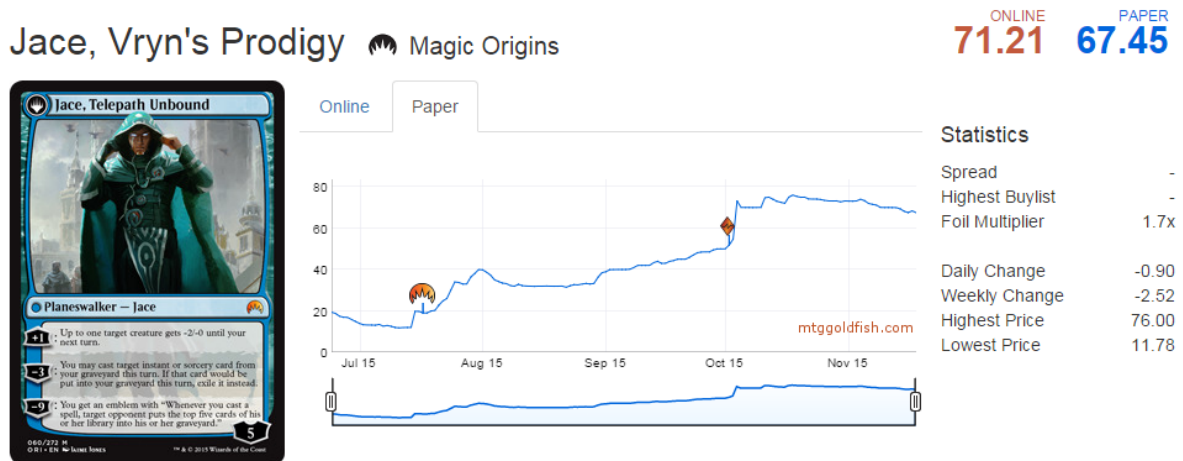
1.1 Purpose

The purpose of this document is to specify and outline requirements of the “mtgCardWallet” development process. It will highlight key decisions and illustrate the development process behind this application. It will distinguish between Functional and Non-functional Requirements and provide an overall – high level description of “mtgCardWallet”, how Users interact and use the system.

It is primarily intended as a proposal to an Investor or Customer to receive the approval for the development.

1.2 Scope

“mtgCardWallet” is a supporting Web Application that is built upon the success of the trading card game Magic: The Gathering with over 15440 individual (Wizards, 2015) that fluctuate in price, similar to stocks as seen in the image below.



(mtggoldfish.com, 2015)

“mtgCardWallet” will allow a User to organise and maintain his trading card collection more efficiently and profit on the fluctuations on the market by integrating “mtgCardWallet” with Europa’s leading sales and purchase platform of individual trading cards “mtgcardmarket.eu” on an affiliate base.

Once a user’s registers his Account on “mtgCardWallet” he will be able to Login and create his collection. The application will provide an updated Database that includes all data related to each individual Magic the Gathering card and each Set to enable a smooth creation process of the collection.

The User provides the name, version, condition of the card, how many copies and any additional data that is required to determine the value of the card such as language or premium. Once this process is completed for the whole collection “mtgCardWallet” will provide the following main services.

- Details on Personal Collection
- Daily Update of Collection Value and individual cards
- Information of Recent Trades
- Creation of Tournament Decks
- Card Wish List

- Easy purchase of missing cards using magiccardmarket API
- Easy sale of collection using magiccardmarket API


In addition to these main features, several additional features will be provided in a later stage. Such as:

- Top Movers and Shakers
- MTG Big Data Analytics on Collection
- Suggestion for collection
- Feedback on played Tournaments
- Tournament Scheduler
- Recent Tournament Deck lists
- Available cards for recent Tournament Decks
- Deck suggestions based on availability of cards

The Application will be funded by participating in the affiliate Program for magiccardmarket and based on Donations. There are options to enter additional markets and provide same services for other successful trading card games such as Pokémon, Star Wars etc. There are currently no plans to implement advertisement. Possible competitors are: mtgprice.com and deckbox.org but none of those are providing the same service as “mtgCardWallet”.

1.3 Abbreviations and Definitions

Term	Definition
User	Someone who interacts with the mtgCardWallet application.
Magic: The Gathering	Magic: The Gathering (MTG; also known as Magic) is a trading card game created by Richard Garfield.
Card	<p>A card is a physical playing card with the below properties:</p> <ul style="list-style-type: none"> • name • manaCost • cmc • colors • type • supertypes • types • subtypes • rarity • flavor • artist • number • power • toughness • layout • imageName • id

	 <p>The image shows a Magic: The Gathering card titled "Jace, Telepath Unbound". It is a blue Planeswalker card with a mana cost of 5. The card features three abilities: a +1 ability that gives a target creature -2/-0 until the next turn; a -3 ability that allows casting a target instant or sorcery card from the graveyard this turn, exiling it instead; and a -9 ability that gives an emblem which causes the target opponent to put the top five cards of their library into their graveyard whenever a spell is cast. The card is numbered 060/272 M and is from the Oracle set.</p>
Format	<p>MTG is played in a vast amount of different formats. The most common is Standard. A format provides rules on what cards are allowed in tournament environments. Standard for example consists of 60 card decks and no card other than basic land card can be included more than 4 times. In addition to that, only the most recent Sets are allowed. Highlander is another format where the deck consists of exactly 100 cards and no card other than basic land can be included more than once. Other formats are Modern, Limited Sealed, Draft, Legacy, Vintage and Two Headed Giant each with individual rules.</p>
Deck	<p>A Deck is a subset of a collection that is designed to be legal in a specific format.</p>
Trade	<p>An exchange of ownership of cards between trading card collectors.</p>
MKM	<p>Short for magiccardmarket.com – Leading provider in Europe</p>
Wizards of the Coast.	<p>Publisher of Magic: The Gathering</p>
Collection	<p>All cards owned by a User</p>
API	<p>An Interface for the application to exchange Data with other websites.</p>
TAG	<p>A unique identifier.</p>
GIST	<p>A description of the concept.</p>
SCALE	<p>Scale of Measure.</p>
METER	<p>Process or device used to establish Scale.</p>
MUST	<p>Minimum Requirement.</p>
PLAN	<p>Successful Requirement.</p>
WISH	<p>Optimal Requirement.</p>
DEFINED	<p>A Definition.</p>

1.4 Overview

The remaining points of this document will include the Overall Description that will highlight the general System and the main features of “mtgCardWallet”, it will also detail the Functional and Non-functional Requirements that will contain all of the testing and quality requirements for “mtgCardWallet” and it will also provide a detailed description of the system and all of its features.

Different specification techniques have been used to specify the requirements in a more precise method to accommodate for different audiences.

The Overall Description

This Overall Description will highlight and provide an overview of the “mtgCardWallet” System. The System will be detailed in its context and will show how the system interacts with other systems and the functionalities of them. Additionally it will provide an analysis of the different stakeholders on how each of them interacts with the system and what different functionalities are available for each individual stakeholder. At the end of this section we will present constraints and assumptions of the “mtgCardWallet” System.

2.1 Product Perspective

The “mtgCardWallet” System consist on an ideal basis of three parts. The Database that contains all details about the trading Cards, User Information and the Data related to each individual collection. The Web Application used as the primary interface for Users to interact with the System with its simplistic User Interface and the API’s of several Web Services to enhance the user experience of “mtgCardWallet. There are currently different solutions on the Administration of “mtgCardWallet” and the decision has not been made at this point. There are options to administer the Site directly using the User Interface by setting up Administrator credentials or to take the administration of the System offline and separate it from User decisions. The final decision will be made in the 3rd Sprint of the Development

The Diagram in this section highlights how the individual parts of the System interact with each other.

Since this System follows a data-centric approach to its implementation the Collection, Users, Decks and Cards have to be centralized and permanently stored within the System. .

For that, a database will be used. Both the mobile application and web portal will communicate with the

A relational Database will be used to centralize all the Data required by different parts of the System, Interfaces have to be established to Read, Write, Delete and Edit Data within the Database directly by Interacting with the User Interface on the Web Application. Security mechanics have to be put in place to ensure those interfaces only serve its purpose

2.2 User Classes and Characteristics

There will be up to 3 different types of users that interact with the System.

- Unregistered Users
- Registered Users
- Administrators

Each Individual User will interact differently with the system and will have different requirements and needs.

The Unregistered User will have to no access to the functionality of the application and only read access to FAQs, Tutorials and Reviews. In order to create a collection the Registration step has to be completed.

Registered Users will have all functionalities available to them. They can pick cards and enter them in their collection, modify it, add trades, they can see the total value of their collection, create decks, create wish list, buy cards from mkm etc. Registered Users are what the application is designed to serve.

As mentioned in 2.1 the final decision on the use of Administrators has not been made yet however, in all cases they should be able to maintain the Databases and ensure the MTG Card / Set Database is up to date and includes the latest Data provided by wizards of the coast. To ensure this task they will have additional permissions and functionalities available to them.

2.3 Constraints

A constraint of the System will be Internet access. All of its parts are not on the same physical location and need to be permanently connected on the World Wide Web. It is critical for all involved parties (User + System) that a connection is established.

The capacity of the Database is another constraint as Users will generate a large amount of records that will all be stored within the same collections table.

The Framework of the Web Application will be generated using Ruby Rails.

2.4 Operating Environment

The Application will be able to run on every modern Internet Browser such as Google Chrome, Mozilla Firefox and Internet Explorer.

2.5 Assumptions and Dependencies

We assume that "mtgCardWallet" will serve European Users primarily in English as it is highly connected with the leading European Magic platform "magiccardmarket.eu". A success in the European market could lead to integration with the leading US platform "TCG player. This is also true for additional language support. It is also assumed that all Users have access to Touchscreen or Mouse and Keyboard.

Requirements

3.1 Functional Requirements

3.1.1 Functional requirement 1.1

ID:	FR1
Title:	Publicly access "mtgCardWallet" on the WWW.
Description:	Any user should be able to access the Website using the public available URL.
Rational:	In order for a user to access the application.
Dependencies:	None
Priority:	A

3.1.2 Functional requirement 1.2

ID:	FR2
Title:	User registration - Web application
Description:	Given that a user has accessed the Web Portal, the user should be able to register through the Interface. The user must provide password and a unique e-mail address.
Rational:	In order for a user to register on the Web application.
Dependencies:	FR1
Priority:	A

3.1.3 Functional requirement 1.3

ID:	FR3
Title:	Retrieve password
Description:	Given that a user has registered, he should be able to retrieve his/her password by email.
Rational:	In order for a user to retrieve his/her password.
Dependencies:	FR1 and FR2
Priority:	A

3.1.4 Functional requirement 1.4

ID:	FR4
Title:	User log-in
Description:	Given that a user has registered by providing a unique email and a secure password, the user then should be able to log in to "mtgCardWallet". The

	log-in information can be stored in the browser cache to allow the user to be logged in automatically.
Rational:	In order for a user to gain access to the functionalities.
Dependencies:	FR1 and FR2
Priority:	A

3.1.5 Functional requirement 1.5

ID:	FR5
Title:	Add card to collection
Description:	Given that the user has logged in successfully. The user should be able to add individual cards to his collection by selecting the add card feature and providing the card name and specify it's set. Additional properties such as promo, language and condition will also be provided.
Rational:	In order for a user to add cards into his collection.
Dependencies:	FR4
Priority:	A

3.1.6 Functional requirement 1.6

ID:	FR6
Title:	Add card to collection – suggestion
Description:	Given that the user is in the middle of FR5. A card suggestion can be made after the user enters the first 3 characters of the card name. The suggestion will include Set and Full card name. By clicking on the suggestion, the card is added to the user's collection after setting the additional properties.
Rational:	In order for a user to increase user experience during FR5.
Dependencies:	FR5
Priority:	B

3.1.7 Functional requirement 1.7

ID:	FR7
Title:	Add card to collection – bulk
Description:	Given that the user has logged in successfully. The user should be able to make a bulk request to add many cards into his collection by using the add bulk by set feature and selecting the set name from the dropdown. All cards of the selected set will be provided and the user will be able to select the cards he wants to add into his collection and specify the additional properties of the individual cards.
Rational:	In order for a user to bulk add cards into his collection.
Dependencies:	FR4
Priority:	B

3.1.8 Functional requirement 1.8

ID:	FR8
Title:	Remove card in collection
Description:	A user should be able to remove cards added to his collection.
Rational:	In order for a user to remove cards from his collection.
Dependencies:	FR5 or FR7
Priority:	A

3.1.9 Functional requirement 1.9

ID:	FR9
Title:	Edit card in collection
Description:	A user should be able edit the cards in his collection and modify the additional properties.
Rational:	In order for a user to show he likes that Snippet
Dependencies:	FR5 or FR7
Priority:	B

3.1.10 Functional requirement 1.10

ID:	FR10
Title:	Get Card Value – current
Description:	The system should be able to make a request to magiccardmarket to retrieve the card value using the additional properties.
Rational:	In order to provide the value of the card in users collection.
Dependencies:	FR5 or FR7
Priority:	B

3.1.11 Functional requirement 1.11

ID:	FR11
Title:	Collection Value – current
Description:	The System should be able to provide the total collection value of the users collection at this point in time by adding all individual cards and calculating a total.
Rational:	In order to provide the total collection value.

Dependencies:	FR3 or FR5
Priority:	B

3.1.12 Functional requirement 1.12

ID:	FR12
Title:	Collection Value – Historical
Description:	The System should be able to store the total collection value of each collection within the system and provide a historical evolution graph.
Rational:	In order to generate an evolution chart of the users collection.
Dependencies:	FR3 or FR5 and FR10
Priority:	C

3.1.13 Functional requirement 1.13

ID:	FR13
Title:	Create Wish list
Description:	The User should be able to select cards for his wish list using an identical process to FR3.
Rational:	In order to purchase cards from mkm.
Dependencies:	FR3 or FR5 and FR10
Priority:	A

3.1.14 Functional requirement 1.14

ID:	FR14
Title:	Purchase Cards from Wish list
Description:	The User should be able to select cards from his wish list and make a API request to put them directly into his shopping card on mkm.
Rational:	In order to purchase cards from mkm.
Dependencies:	FR3 or FR5 and FR10
Priority:	A

3.1.15 Functional requirement 1.15

ID:	FR15
Title:	Add Deck to collection
Description:	The User should be able to add a Deck to his collection by choosing a format and a deck name.
Rational:	In order to organize the collection.
Dependencies:	FR3 or FR5
Priority:	B

3.1.16 Functional requirement 1.16

ID:	FR16
Title:	Add Card to decks
Description:	The User should be able to select cards from his collection that satisfy the restrictions on the format on the deck and add them into his collection.
Rational:	In order to organize the collection.
Dependencies:	FR15
Priority:	B

3.1.17 Functional requirement 1.17

ID:	FR17
Title:	Display Deck Information on Collection Overview
Description:	The System should be able to display information about the deck a card is currently in.
Rational:	In order to organize the collection.
Dependencies:	FR16
Priority:	C

3.1.18 Functional requirement 1.18

ID:	FR18
Title:	Add trades to collection
Description:	A user should be able to add a trade to his collection by selecting the option in the menu. The user chooses a name, cards he received and cards he traded out of his collection. The System then should make automatic updates on his collection.
Rational:	In order to maintain the collection.
Dependencies:	FR5
Priority:	C

3.1.19 Functional requirement 1.19

ID:	FR19
Title:	Display all trades by user
Description:	The System should be able to list all trades and all details surrounding the trade to the user. This should provide update on pricing and display information about the deck a card is currently in.
Rational:	In order to provide useful information to the user.

Dependencies:	FR18
Priority:	C

3.1.20 Functional requirement 1.20

ID:	FR20
Title:	Filter results of FR19 (trades by user)
Description:	The System should be able to provide filter options to the user.
Rational:	In order to enhance experience of FR19.
Dependencies:	FR19
Priority:	C

3.1.21 Functional requirement 1.21

ID:	FR21
Title:	Display most recent tournament decks – Data
Description:	The System should be able to display deck lists of the most recent decks gathering them from starcitygames and other leading online providers.
Rational:	In order to add functionality to the System.
Dependencies:	None
Priority:	D

3.1.22 Functional requirement 1.22

ID:	FR22
Title:	Provided Details on Individual Cards
Description:	The System should be able to display details about individual cards including the information on recent tournament decks that included the card.
Rational:	In order to increase user experience.
Dependencies:	FR21
Priority:	D

3.1.23 Functional requirement 1.23

ID:	FR23
Title:	Display % of cards owned that are part of a tournament deck.
Description:	The System should be able to display a % behind each tournament deck that shows the User the percentage of cards included in his collection.
Rational:	Enhance User experience
Dependencies:	FR22
Priority:	D

3.1.24 Functional requirement 1.24

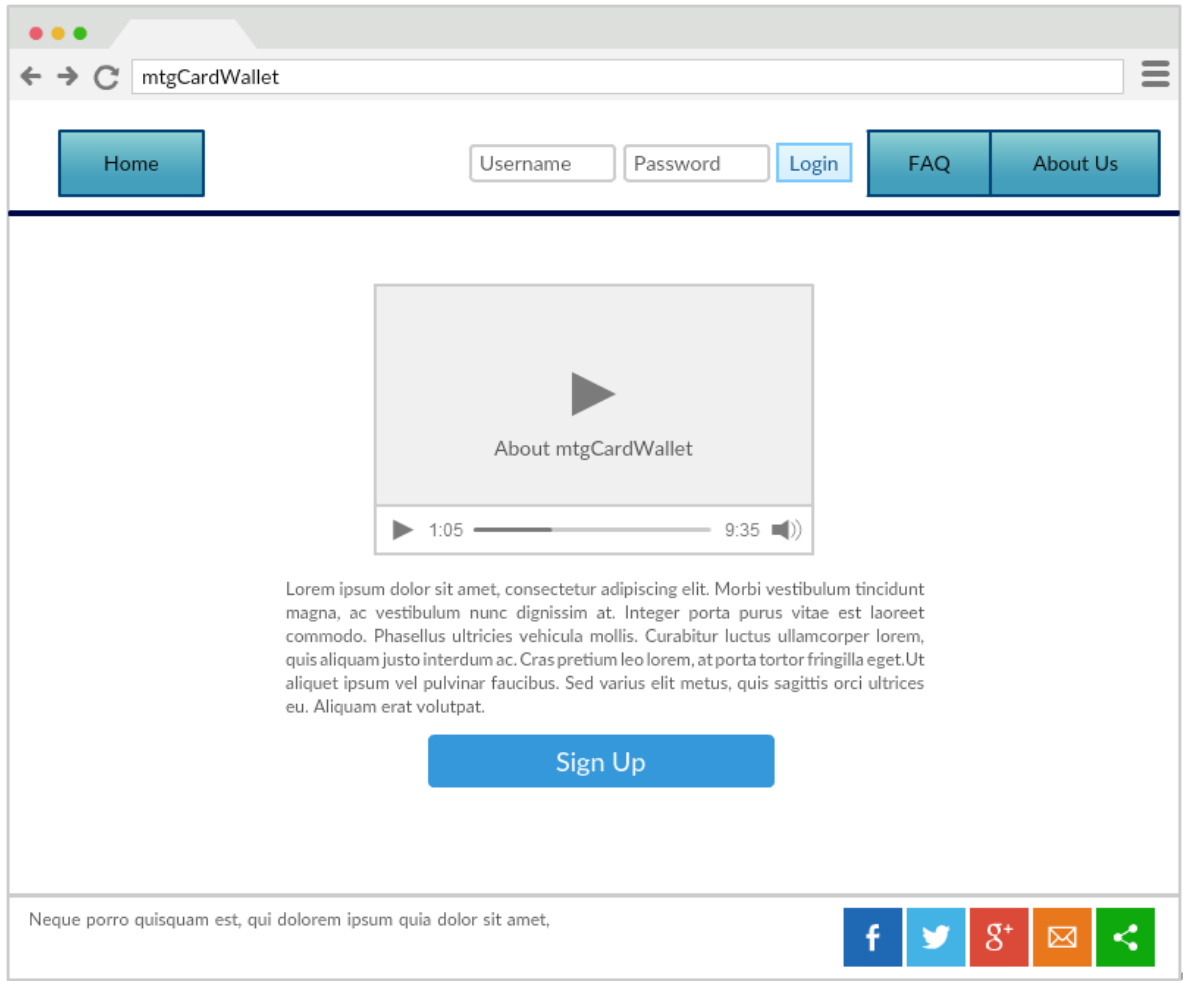
ID:	FR24
Title:	Add missing cards of FR23 to wish list.
Description:	The System should be able to add the missing cards of the user's collection into his wish list.
Rational:	In order to create revenue
Dependencies:	FR23 and FR13
Priority:	D

3.2 External Interfaces

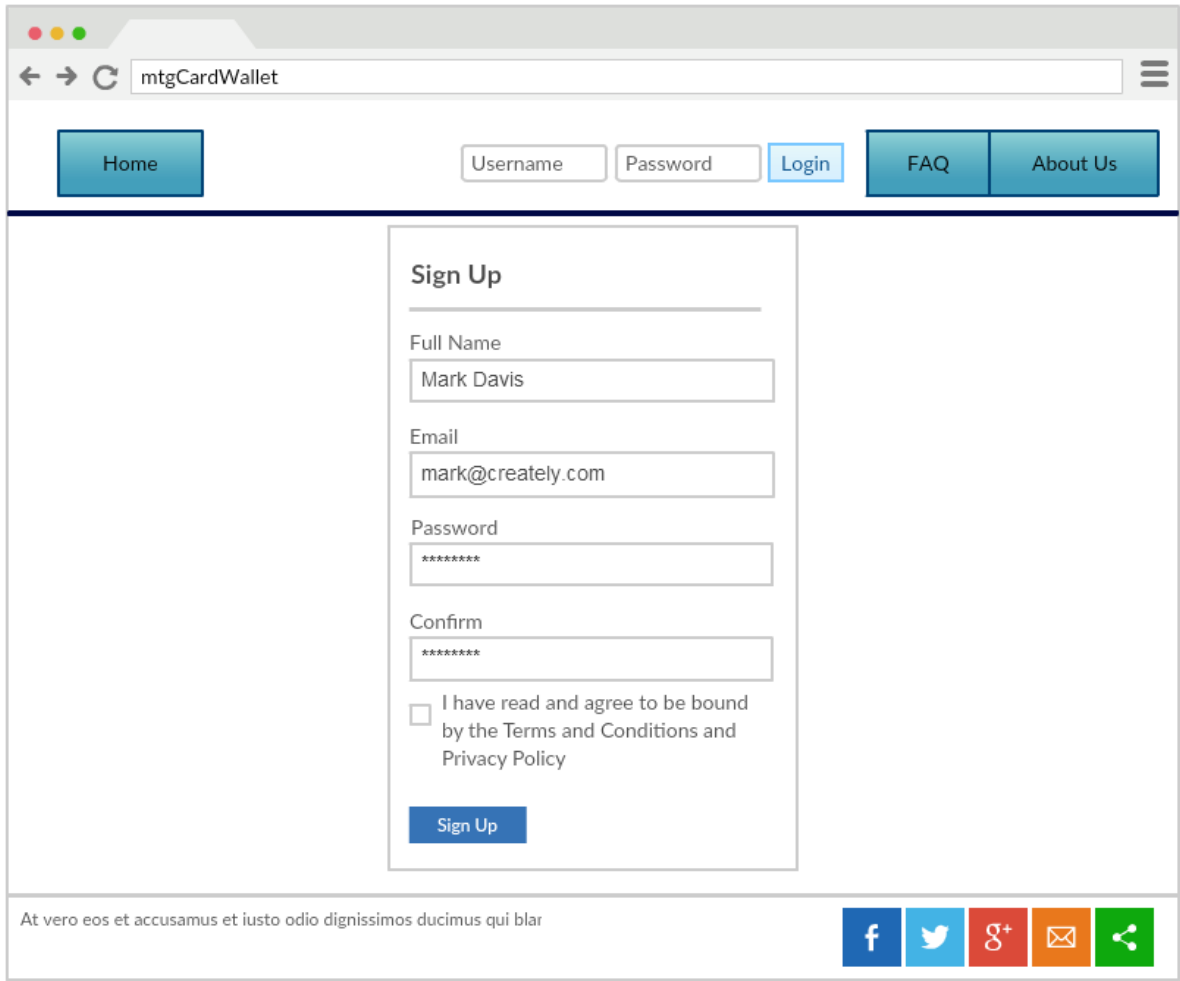
The external Interfaces list the inputs and outputs of the system and provide a description of the software, hardware components and communication interfaces. It also provides a basic prototype of the User Interface.

3.2.1 User Interface

Accessing "mtgCardWallet" via the Web will lead the User to the Home Page. It will provide a high level description of the application, it's features and an FAQ. An already registered user will be able to login to the site by entering a username and password in the fields provided. New Users will be able to Sign up for "mtgCardWallet". This Site will also display sharing features in the footer of the page.



Clicking on the Sign Up Button on the middle of the page will direct the user to the Account creation task.



After the user signed up successfully for the Site he will be able to login. A successful login will direct the user to the collection page.

mtgCardWallet

Home Username | Collection Value (Eur) | Log Out Collection Decks Wish list

Top Winners	%	\$	Top Losers	%	\$
Tarmogoyf	10%	10\$	Tarmogoyf	-10%	-10\$
Dark Confidant	1%	1.25\$	Dark Confidant	-1%	-1.25\$
Jace	0.5%	0.40\$	Jace	-0.5%	-0.40\$
...			...		
..			..		
.			.		

1820\$ (+57\$)

Add Edit

Cardname	Language	Condition	Set	Foil	Signed	Playset	Altered	Value
Tarmogoyf	EN	NM	MM2015	TRUE	FALSE	TRUE	FALSE	1000\$ (250\$)
Dark Confidant	EN	NM	MM2014	TRUE	FALSE	TRUE	FALSE	500 (125\$)
Jace	DE	Mint	Origins	FALSE	FALSE	FALSE	FALSE	320\$ (80\$)
...								
..								

sum lectus eget justo. Vivamus venenatis semper mauris eu auctor.

f t g+ e s

The Collection Site will display the current collection and allow the User to add and Edit it. It will highlight the top Movers and Shakers to provide more insight into the collection of the signed in User.

mtgCardWallet

Home Username | Collection Value (Eur) | Log Out Collection Decks Wish list

[New](#) [Edit](#)

Deck 1	Format
Deck 2	Format
Deck 3	Format 2


Main:

Amount	Cardname
3	Liliana of the Veil
4	Tarmogoyf
4	Dark Confidant
2	Scavenging Ooze
2	Kitchen Finks
1	Huntmaster of the Fells
1	Tasigur the Golden
3	Inquisition of Kozilek
3	Thoughtseize
1	Blood Crypt
...	
..	
.	
60 TOTAL	

Sideboard:






Amount	Cardname
3	Liliana of the Veil
4	Tarmogoyf
4	Dark Confidant
2	Scavenging Ooze
...	
..	
.	
15 TOTAL	

[Print](#) [Statistic](#)

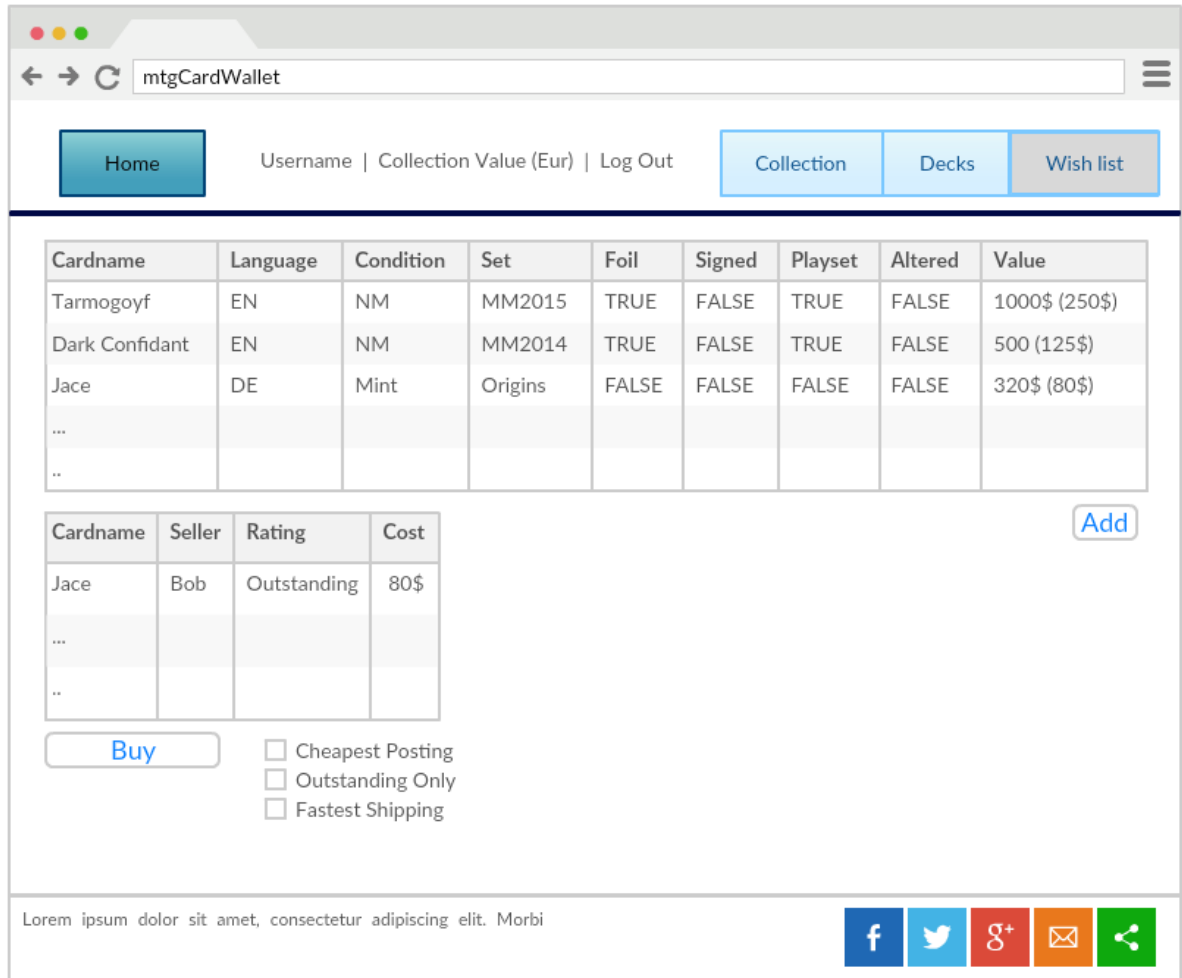


Statistic
WIN | LOSE | %

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accus.

The Decks tab lists all decks of the User and additional statistics. This Site is used to maintain the Decks by providing an "edit" and "new" interface.



The Wish List tab is connected with the API for macigcardmarket and allows the user to add cards into his collection and buying them directly within the application. An affiliate agreement with macigcardmarket has been arranged and a small % of the revenue will be added.

3.2.2 Hardware Interface

None of our application the web portal or mobile application have any designated hardware requirements as it does not have any specific hardware interfaces.

3.2.3 Software interfaces

The Web application communicates with the Database in order to fetch information about Cards, Decks, Wish List, API etc. The communication between the database and the portal consists of both reading and modifying data.

3.3 Nonfunctional Requirements

The requirements outlined in this section specify clearly the required reliability, availability and security of “mtgCardWallet”.

3.3.1 Performance requirements

ID:	PR1
TAG:	ResponseTime
GIST:	The speed of the Database response.
SCALE:	Response time of a request.
METER:	Obtained from 100 Requests during testing.
MUST:	No more than 2 seconds 98% of the time.
WISH:	No more than 1 second 100% of the time.

ID:	PR2
TAG:	SystemAvailability
GIST:	Fault tolerance of “mtgCardWallet”.
SCALE:	If the Server is unavailable or a large Queue on process requests, the user should be informed about it.
METER:	Obtained from 10 hours during testing.
MUST:	100% of the time.

ID:	PR3
TAG:	InternetConnection
MUST:	100% of the time.

3.3.2 Security requirements

ID:	SR1
TAG:	AccountSecurity
GIST:	Security of all user accounts.
SCALE:	If a User tries to log in to the web portal with a non-existing account then he should not be logged in.
METER:	10 attempts to log-in with a non-existing user account during testing.
MUST:	100% of the time

ID:	SR2
TAG:	DataCommunication
GIST:	Security of the communication between “mtgCardWallet” and server
SCALE:	The messages send should be encrypted for log-in communications.
METER:	Attempt to get user-name and password through obtained messages on 10 log-in session during testing
MUST:	0% of all messages.

6.3 User Feedback Form

mtgCardWallet User Feedback

Please fill out this Survey after the Demo and let me know if you have any suggestions.

1. How do you like the design?

Mark only one oval.

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2. How satisfied are you with the User Interface?

Mark only one oval.

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. How satisfied are you with the performance?

Mark only one oval.

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4. How satisfied are you with the reliability?

Mark only one oval.

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. What did you like most about mtgCardWallet?

Check all that apply.

- User Interface
- Card Search
- Life Counter
- Performance
- Ease of Use

6. Would you recommend mtgCardWallet?

Check all that apply.

- Yes
- No

7. Do you have any thoughts on how to improve mtgCardWallet

.....

.....

.....

.....

.....

6.4 Monthly Journals

September 2015

Summary:

As I made the decision to host my site on heroku, they recommend using postgresql so I need to find an efficient solution to create the Database using postgres.

Requirements:

- More than 15.000 individual cards
- Around 50 properties for each individual cards
- More than 100 different Sets
- Around 15 properties for each set.
- Each individual card belongs to 1 set
- Prices for Individual Cards have to be updated daily (fetched from API?)

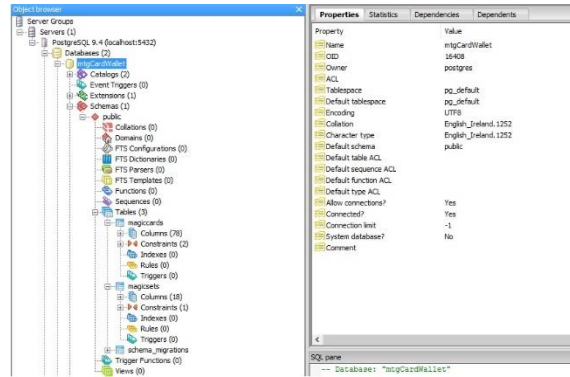
I found a tool Gathering Extractor that extracts all card and set Information from [Gatherer](#) (official mtg card source) and exports it to SQL. They even provided pricing details but based on \$ and only upon initial setup.

Note to self: Could be run every day to implement pricing data based on snapshots.

Completed:

- Build Local PostgreSQL Database using pgAdmin|||
- Established connection between local Database and mtgCardWallet Ruby App
- Insert MTG Cards and Sets into Database based on Gatherer Extractor

Note that only Battle for Zendikar has been added as it's sufficient for testing purpose



Enable Heroku deployment

Heroku has been implemented and the Site is now up and running on:

<http://morning-brushlands-5143.herokuapp.com/>

Enable GIT backup

Repository is online and used for Version control:

<https://github.com/mgravina/mtgCardWallet>

Ongoing:

- Database Design completion
- Create User Model for Login
- Create Login Screen and functionality
- Implement Classes for Bootstrap implementation

Research:

API Implementation research findings.

Best Source of European Pricing Information is magiccardmarket.eu

api -> https://www.mkmap.eu/ws/documentation/API_Main_Page

In order to get the pricing using the mkmap I need to use the version 1.1 or 2.0 which is currently in beta and only available on Sandbox Server -> Created User: Racador on Sandbox server and requested API tokens.

They provide details for enabling OAUTH for C#, VB, PHP but not ruby.

https://www.mkmap.eu/ws/documentation/API:Auth_Overview

Will research Rails and see how to get Oauth to work and make requests that suit my need. Found support on the topic. Will Download App and explore.

Found resources on the topic. Will investigate

<https://github.com/oauth-xx/oauth-ruby>

Notes:

- Create dump file using this command:

```
"C:\Program Files\PostgreSQL\9.4\bin\pg_dump.exe" -Fc --no-acl --no-owner -h localhost -U postgres
mtgCardWallet > mydb.dump
```

- Upload dump file onto Dropbox
- Upload dump file onto heroku DB using below command and change ending of URL to dl=1

heroku pg:backups restore 'INSERT_DROPBOXURL_HERE' DATABASE_URL

October 2015

Summary:

Initially I wanted to continue working on my Database and User Model and started thinking about how to allow rails to look at the database created yesterday and implement a scaffold based on all of those values. I realized quickly that I had a lot of catching up to do and started to go through Chapter 4 on https://www.railstutorial.org/book/rails_flavored_ruby to refresh my memory and read about a few concepts that are helpful and applicable to my application. I also touched on CSS and Bootstraps and in order to not have any further problems down the road I will need to implement the underlying structure. Will add this to my todo list.

Completed:

TDD concept explored and implemented for Application
Testing added to application
Implementation of Ruby "DRY" Don't Repeat yourself! principle
Ruby Code Test and minor code improvements

Ongoing:

CSS class structure
Database Design
Create User Model
Create Login Screen and functionality
Scaffold existing DB entries based on DB Schema

Research:

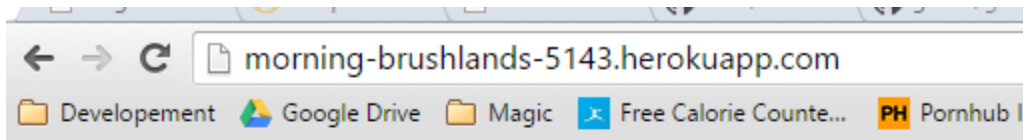
Research has been done to provide an answer to the question. How to efficiently approach controllers and views for existing Database.

My approach at the moment will be:

- Create the User Model in Ruby and apply it to local postgres DB. (rake db:migrate)
- Create other tables and relationships in pgAdmin and create controllers etc automatically.
- Initial research lead me to: <https://www.ruby-forum.com/topic/6875485> Will further investigate and add to todo List.

Notes:

How to delete branch in git:
git branch -d branch1
git push origin :branch1
rails console to open ruby command line
rake test to enable test of changes
Application Snapshot:



Sample App

This is the home page for the [Ruby on Rails Tutorial](#) sample application.

Hashes in Ruby:

```
>> user = {} # {} is an empty hash.
=> {}>> user["first_name"] = "Michael" # Key "first_name", value "Michael"
=> "Michael">> user["last_name"] = "Hartl" # Key "last_name", value "Hartl"
=> "Hartl">> user["first_name"] # Element access is like arrays.
=> "Michael">> user # A literal representation of the hash
=> {"last_name"=>"Hartl", "first_name"=>"Michael"}
```

TDD test case layout:

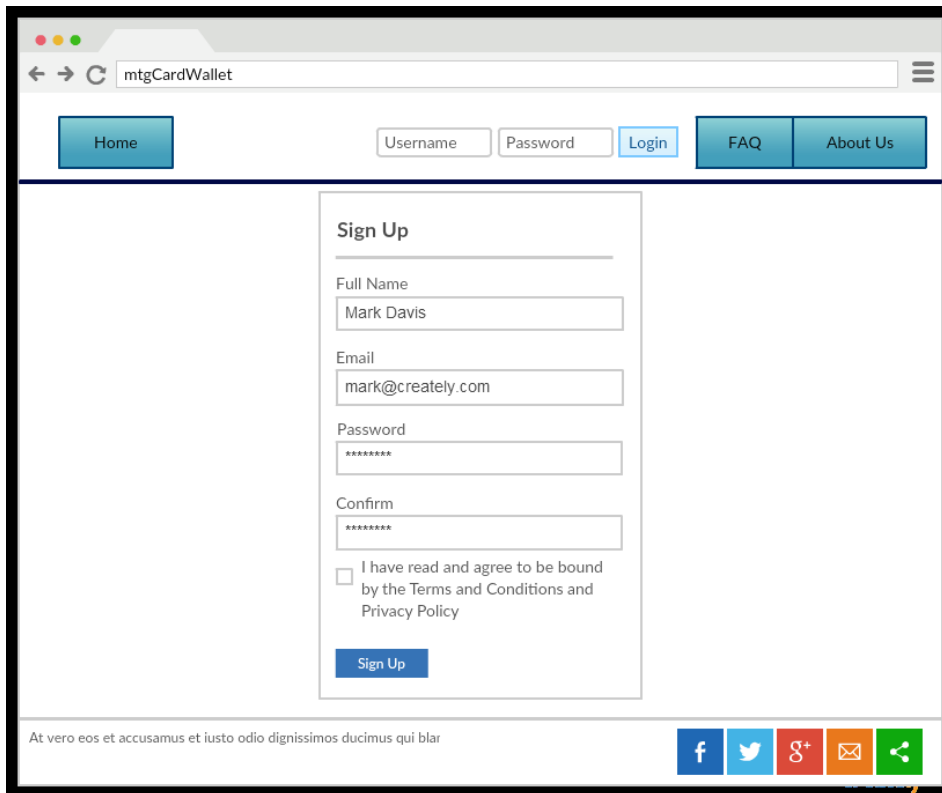
```
test "should get home" do get :home
  assert_response :success
  assert_select "title", "Ruby on Rails Tutorial Sample App"end
```

November 2015

Summary:

I finished the Bootstrap and CSS work. The basic design is finished, after implementing the User Model. I'll start with the Sign Up and Profile Pages.

I started to look into available user models to implement into my application but it became very clear that designing my own will have several benefits. I am able to customize it and make it compliant with all alternative login methods (FB, Google etc). By researching the User Model on ruby I read about the "bcrypt" gem (<https://github.com/codahale/bcrypt-ruby>) that gets around the idea of storing passwords in my database. The user still needs a password but it won't be stored. I'm happy with the implementation and the next step will be designing the Sign Up Screen. I made a mock up and will stick to the simple design.



The Database User model:

users	
id	integer
name	string
email	string
created_at	datetime
updated_at	datetime
password_digest	string

It is very basic and there are a few things still missing:

Foreign Key for Collection Table

Placeholder for alternative login info

Validation:

For the moment I implemented a very basic password validation that only checks for:

#ensure that all entries are saved as lowercase

```
before_save { self.email = email.downcase }
```

```
validates :name, presence: true, length: { maximum: 50 }
```

```
VALID_EMAIL_REGEX = /\A[\w+\-\.]+\@[a-z\d\-\.\+]\.[a-z]+\z/i
```

```
validates :email, presence: true, length: { maximum: 255 },
```

```
  format: { with: VALID_EMAIL_REGEX },
```

```
  uniqueness: { case_sensitive: false }
```

```
has_secure_password
```

validates :password, length: { minimum: 6 }

This will be reworked in a later version.

I increased the total number of TestCase and continued on my TDD.

```
-----
UserTest: test_email_validation_should_reject_invalid_addresses
-----
User Exists (0.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user@example.com') LIMIT 1
User Exists (15.8ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user_at_foo.org') LIMIT 1
User Exists (0.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user.name@example.') LIMIT 1
User Exists (0.7ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('foo@bar_baz.com') LIMIT 1
User Exists (1.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('foo@bar+baz.com') LIMIT 1
(0.0ms) ROLLBACK
(0.0ms) BEGIN===== ] 73% Time: 00:00:00, ETA: 00:00:00
-----
UserTest: test_name_should_be_present
-----
User Exists (1.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user@example.com') LIMIT 1
(0.0ms) ROLLBACK
(0.0ms) BEGIN===== ] 80% Time: 00:00:00, ETA: 00:00:00
-----
UserTest: test_password_should_be_present_(nonblank)
-----
User Exists (0.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user@example.com') LIMIT 1
(0.0ms) ROLLBACK
(0.0ms) BEGIN===== ] 86% Time: 00:00:00, ETA: 00:00:00
-----
UserTest: test_password_should_have_a_minimum_length
-----
User Exists (0.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user@example.com') LIMIT 1
(0.0ms) ROLLBACK
(0.0ms) BEGIN===== ] 93% Time: 00:00:00, ETA: 00:00:00
-----
UserTest: test_should_be_valid
-----
User Exists (0.0ms) SELECT 1 AS one FROM "users" WHERE LOWER("users"."email") = LOWER('user@example.com') LIMIT 1
(0.0ms) ROLLBACK
15/15: [=====] 100% Time: 00:00:00, Time: 00:00:00
-----
Finished in 0.63224s
15 tests, 33 assertions, 0 failures, 0 errors, 0 skips
19:46:26 - INFO - Run 'gem install win32console' to use color on Windows
[2];[Minitest results] 15 tests
```

Completed:

- TDD concept explored and implemented for Application
- Testing added to application
- Implementation of Ruby "DRY" Don't Repeat yourself! principle
- Ruby Code Test and minor code improvements
- Create User Model
- CSS class structure

Ongoing:

- Database Design
- Create Login Screen and functionality
- Scaffold existing DB entries based on DB Schema

Research:

Research has been done on Regex and how to provide a proper validation.

Notes:

To check RegEx on Ruby use: <http://www.rubular.com/>

Expression	Meaning
<code>/\A[\w+\-\.]+\@[a-z\d\-\.]+\.[a-z]+\z/i</code>	full regex
<code>/</code>	start of regex
<code>\A</code>	match start of a string
<code>[\w+\-\.]+\</code>	at least one word character, plus, hyphen, or dot
<code>@</code>	literal "at sign"
<code>[a-z\d\-\.]+\</code>	at least one letter, digit, hyphen, or dot
<code>\.</code>	literal dot
<code>[a-z]+\</code>	at least one letter
<code>\z</code>	match end of a string
<code>/</code>	end of regex
<code>i</code>	case-insensitive


Application Snapshot:


mtgCardWallet Home My Collection Log in

Welcome to the mtgCardWallet

This is the home page for mtgCardWallet

[Sign up now!](#)



Your collection in your wallet by Michele Gravina 

[About](#) [Contact](#) [News](#)

December 2015

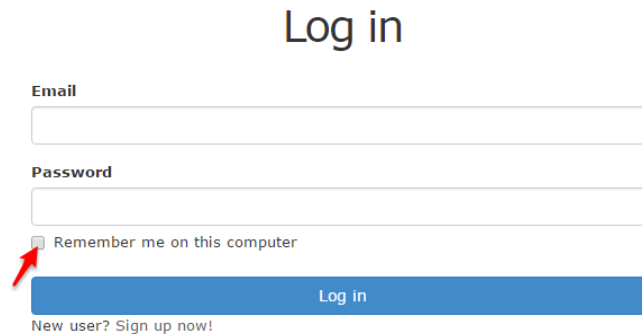
Summary:

Login / Sign Up:

The next step of the completed user model was to create the Sign Up and Login functionality. I based it on the earlier implementation of bootstrap but kept it simple for now.

Login Screen:

*Note that Login functionality within header added to todo list.

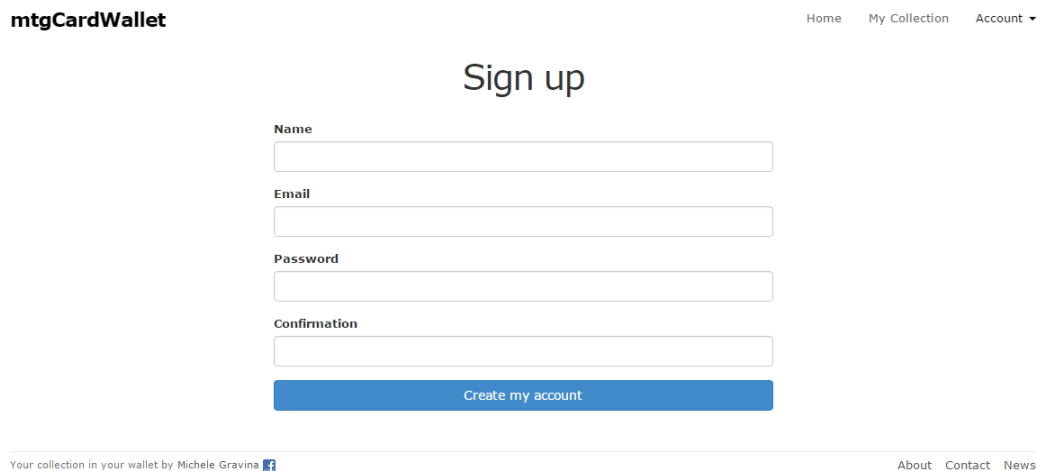


The screenshot shows a login form with the following elements:

- Header: "Log in"
- Form fields: "Email" and "Password", each with a corresponding input box.
- Checkbox: "Remember me on this computer" with a red arrow pointing to the checkbox.
- Button: "Log in" (blue button)
- Text: "New user? Sign up now!"

As you can see in the above Login Screen I also implemented session information and cookies. Highlighted by the "Remember me". If the checkbox is ticked a cookie is created and stored within the browser.

Sign Up:



The screenshot shows a sign up form with the following elements:

- Header: "mtgCardWallet" (left) and "Home My Collection Account" (right)
- Section Header: "Sign up"
- Form fields: "Name", "Email", "Password", and "Confirmation", each with a corresponding input box.
- Button: "Create my account" (blue button)
- Footer: "Your collection in your wallet by Michele Gravina" (left) and "About Contact News" (right)

Security & Encryption:

As we covered Encryption and Security in the Business and Network Security module I looked at my application and decided to implement a few common strategies to secure the Application as much as possible.

I implemented the Secure Socket Layer (SSL) on the whole application. This ensures that all Data send within the application is secure. Second, I looked for good technologies for passwords and implemented

added a gem called "bcrypt". This System ensures that no Passwords or other sensitive Data is stored in my Database.

How bcrypt works:

A simple explanation is that i'm storing the result of a mathematical expression within my database instead of the password. A user enters the password, it's send through the bcrypt and a result string is generated. The result string is stored in my DB. Whenever a user log's in. The result string is checked and if they match. The user us allowed to log in.

User Account Edit & Authorization

With the User Account created completed I had to allow the user to edit their own details. With this implementation done, the next step was Authorization so the user can only edit his own page: This is done using before_action:

```
+class MycollectionsController < ApplicationController  
+ before_action :logged_in_user  
+ before_action :correct_user
```

I'm doing 2 things here. 1st I ensure that only logged in users are able access the edit page and the 2nd is that you can only access your own page.

The same implementation is done for "MyCollection" and will also be done for "Decks".
*It's likely that I want people to share the collection with friends, therefor I might add an additional functionality of public and private or share later.

Edit Account Settings:

Account Settings

Name

Email

Password

Confirmation

[Save changes](#)

Member since: January 10, 2016

I increased the total number of TestCase and continued on my TDD.

Completed:

- Create Login Screen and functionality
- Security and Encryption
- Authorization

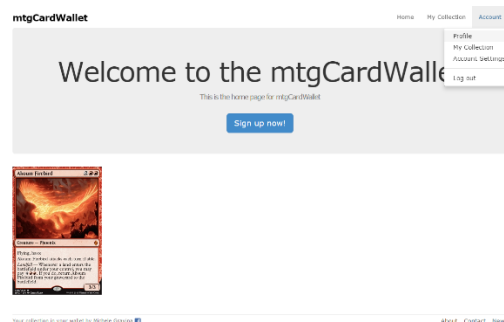
Ongoing:

- Database Design
- Scaffold existing DB entries based on DB Schema

Research:

Research has been done on encryption methods, session information and cookies.

Application Snapshot:

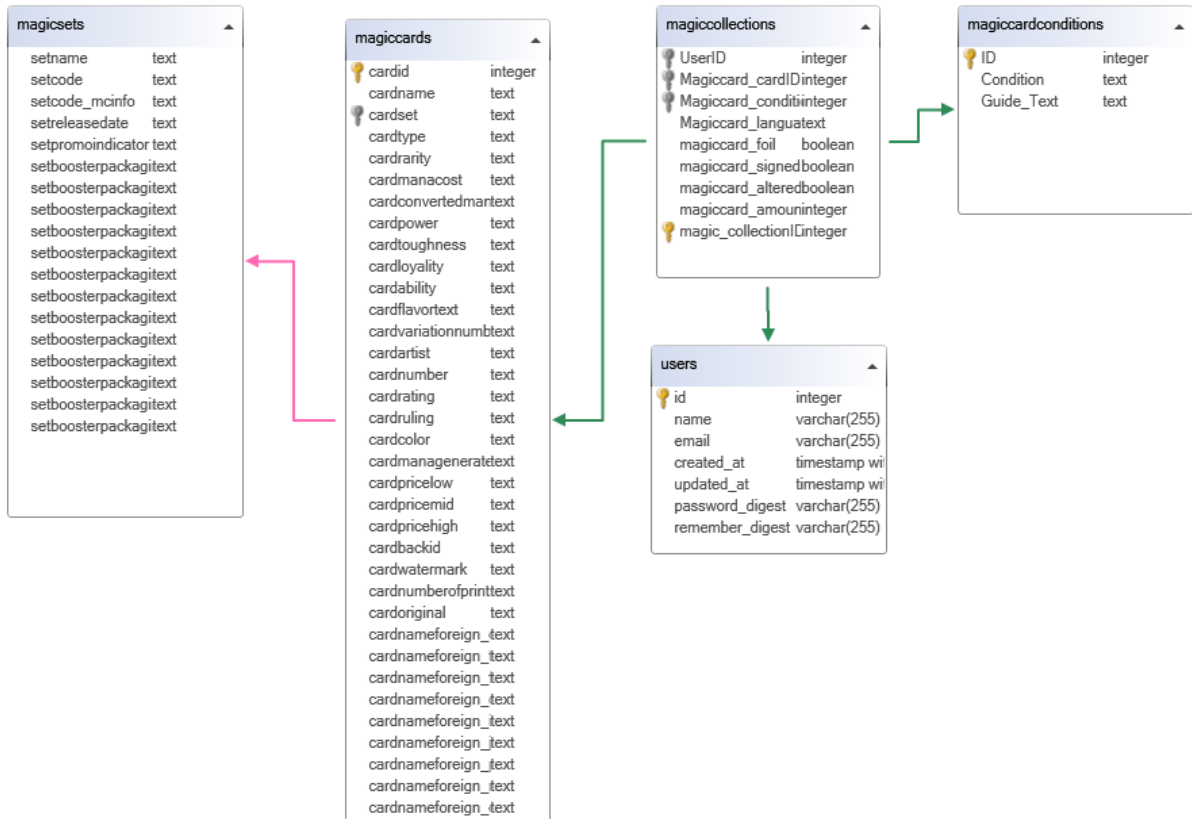


January 2016

Summary:

Database Design:

The first version of the Database has been designed and implemented.



The design is not optimal and a few flaws have been identified in the implementation.

- User has_one magiccollection
- magiccollection has_many magiccards
- magiccards belong_to one or more magicsets
- magicconditions belongs_to magiccards within magiccollection

The flaw identified is:

magiccards belong to many magiccollection <-- I need to normalize and suggest come up with a different structure.

ActiveRecord:

In Order to reference the Database within my Application ActiveRecord is the correct way to handle it, below is an example of the relationships of the Magic_collector. As everything is inherited by ActiveRecord::Base

it's no problem to reference them within the program:

```
class Magic_collection < ActiveRecord::Base
  belongs_to :magiccards
  belongs_to :user
end
```

How to reference?

<%= @user.created_at %> <--- this will return the current user created_at entry in the DB.
*Note that Login functionality within header added to todo list.

Completed:

- Create Login Screen and functionality
- Security and Encryption
- Authorization

Ongoing:

- Database Design (1st draft)
- Scaffold existing DB entries based on DB Schema

Research:

ActiveRecord and Database Design: http://guides.rubyonrails.org/association_basics.html

Application Snapshot:

mtgCardWallet

Home My Collection Account ▾

Account Settings


Name

Email

Password

Confirmation

Member since: February 4, 2016

Your collection in your wallet by Michele Gravina 

About Contact News

```
... /ruby/hash:ActionController::Parameters
controller: users
action: edit
id: '41608530'
```

February 2016

Due to the College Work and business travels not much work has been done in February. There is an open argument that mtgCardWallet should be moved to a different platform as Heroku will not be able to support the complexity of the Database. There are certain benefits that come with the move to Android.

ToDo:

Convert Database to SQLite
Convert User Interface to XML

I will be moving to Android.

March 2016

I converted the Database from PostgreSQL to SQLite and also moved the Database from a testing environment into a production environment.

Name	Type	Schema
Tables (6)		
> MTG_cards		CREATE TABLE MTG_cards(MTG_card_id text PRIMARY KEY,MTG_card_name text,MTG_card_set text,MTG_card_type text,MTG_card_rarity t
> MTG_sets		CREATE TABLE MTG_sets(MTG_set_name text,MTG_set_code text PRIMARY KEY,MTG_set_code_magiccards text,MTG_set_date text,MTG_se
> android_metadata		CREATE TABLE "android_metadata" ("locale" TEXT DEFAULT 'en_US')
> collections		CREATE TABLE "collections" (`_id` INTEGER, `MTG_card_id` text, `card_language` text, `condition` text, `foil` NUMERIC, `signed` boolean, `altered` boolean, `quantity` integer, PRIMARY KEY(`_id`), FOREIGN KEY(`MTG_card_id`) REFERENCES `MTG_cards`(`MTG_card_id`))
> decks		CREATE TABLE "decks" (`_id` INTEGER, `deck_name` TEXT, `MTG_card_id` TEXT, `amount_mainboard` INTEGER, `amount_sideboard` TEXT, PRIMARY KEY(`_id`), FOREIGN KEY(`MTG_card_id`) REFERENCES `MTG_cards`(`MTG_card_id`))
> wishlist		CREATE TABLE "wishlist" (`_id` INTEGER, `MTG_card_id` TEXT, PRIMARY KEY(`_id`), FOREIGN KEY(`MTG_card_id`) REFERENCES `MTG_cards`(`MTG_card_id`))
Indices (2)		
sqlite_autoindex_MTG_cards_1		
sqlite_autoindex_MTG_sets_1		
Views (0)		
Triggers (0)		

All cards and set details have been added and i'm now working on a method to add the Database into my Android App.

I found a solution that creates the Database from a db.file.

```
public DataBaseHelper(Context context) {
```

```

        super(context, DB_NAME, null, 1);
        this.myContext = context;
    }

    /**
     * Creates a empty database on the system and rewrites it with your own database.
     * */
    public void createDataBase() throws IOException {

        boolean dbExist = checkDataBase();

        if(dbExist){
            //do nothing - database already exist
        }else{

            //By calling this method and empty database will be created into the
            default system path
            //of your application so we are gonna be able to overwrite that database
            with our database.
            this.getReadableDatabase();

            try {

                copyDataBase();

            } catch (IOException e) {

                throw new Error("Error copying database");

            }

        }

    }
}

```

This requires a few modifications to the Database but it's working.

April 2016

I made some progress on the User Interface and it's layout design however exam time and project work is coming up and I'll continue after.

Submission date is close, I don't know how I'll finish it.