Ben Gurion University

The program in Software Engineering

Collectomania

Final Document

Authors: Yuri Gabaev

Maria Kertsburd

Olga Sibiryov

Amir Eluk

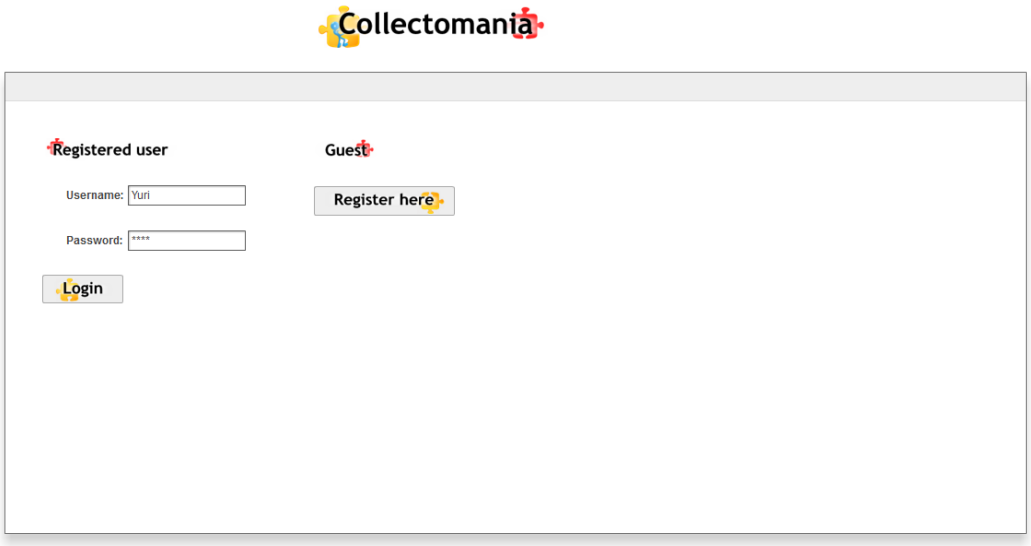
Revision: 1.0

Date: 29 July, 2012

Contents

1. User Manual

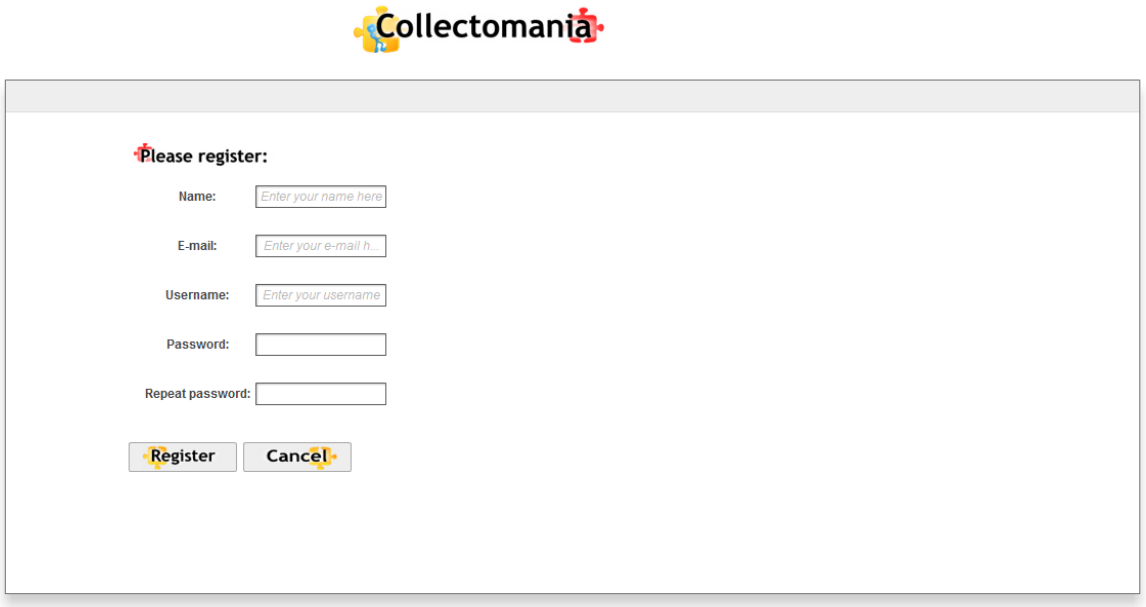
1.1 Login Window



The registered user enters his username and password in the appropriate fields, and clicks on “Login” button, and goes to “Welcome window”

Unregistered user clocks the “Register here” button and goes to “Registration window”.

1.2 Registration Window

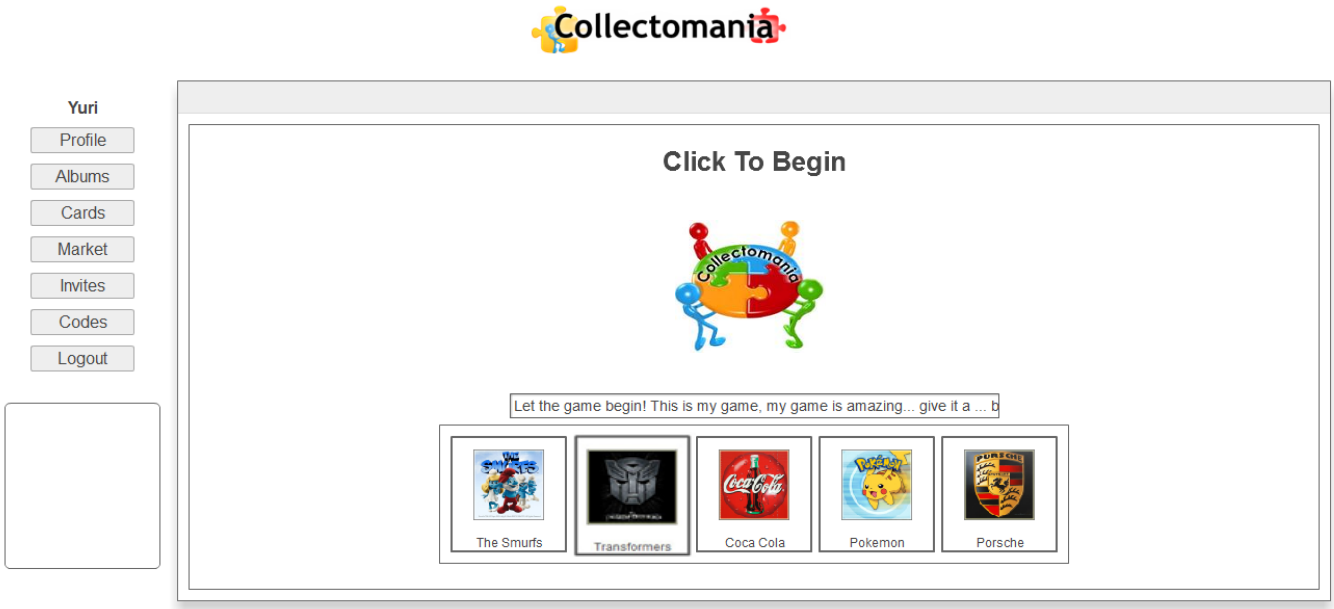


The unregistered user enters his name, e-mail, selected username and password. When he finishes, he clicks the button “Register” and waits for confirmation of registration. When the registration is done, the user goes back to “Login Window”.

If the registration is unsuccessful the user gets message with the specified problem and can fill in the details again.

Of course, the user has an option to cancel the registration at any moment.

1.3 Welcome Window



In the middle of the window the user can see general notifications about the game and list of available albums.

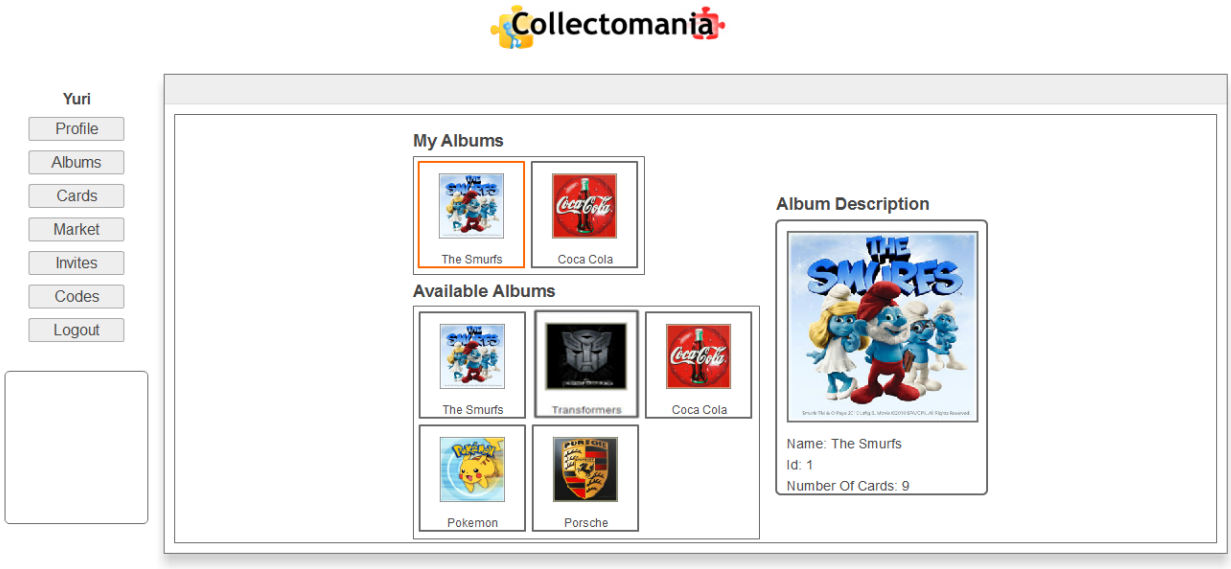
On the left side the user can see available options of windows (albums, cards etc.)

The frame below the menu would be filled with different messages and notifications that are directed to the user. The user can erase a message by clicking on this frame.

When the user clicks on the logo in the center of the page (or when he clicks on the "Albums" button in the left menu), he goes to the "Albums Window".

If the user clicks on "Logout" button, he goes back to the "Login Window".

1.4 Albums Window



In the top list of “My Albums”, the user can see his current albums. If he clicks on the picture of the album he can see some details about the window in the right window (titled with “Album description”. If user clicks twice on the picture of one of his albums, he goes to “Single Album Window”.

1.5 Single Album Window

On this window user can see two pages of his album. He can move between the pages by clicking on “Previous Page” and “Next Page” buttons.

He can see what cards he needs to paste (according to the bright pictures on every page) and the list of his not pasted cards – “My cards” list.

There are several options on this window (The changes are in bold red frames):

a. Information about the card.

By clicking on one of the cards in “My Cards” list, the user gets in the central frame on this page a larger version of the card and some information about this card:

b. Pasting the card:

When user clicks on “Paste card” button, the card is pasted on the appropriate place (if it is shown in the current page, otherwise the user gets an error in the left frame). The picture becomes highlighted and the user gets a notification about the performed action in the left frame:

The picture disappears from the central frame and from the list of not pasted cards.

a. Adding card to wish-list.

By double click on one of the bright pictures on the album’s pages the user can add this card to his wish-list (the list of cards he doesn’t have, but wishes to have).

If the card does not appear in the wish-list, it is added ant the user gets appropriate notification in the left frame. If the card appears in the wish-list the user gets an error notification.

Let's take a look on two examples:

1. The card of the cat was selected (and it appears in the wish-list):

2. The card with id 106 was selected (and it does not appear in the wish-list).

1.6 Cards Window

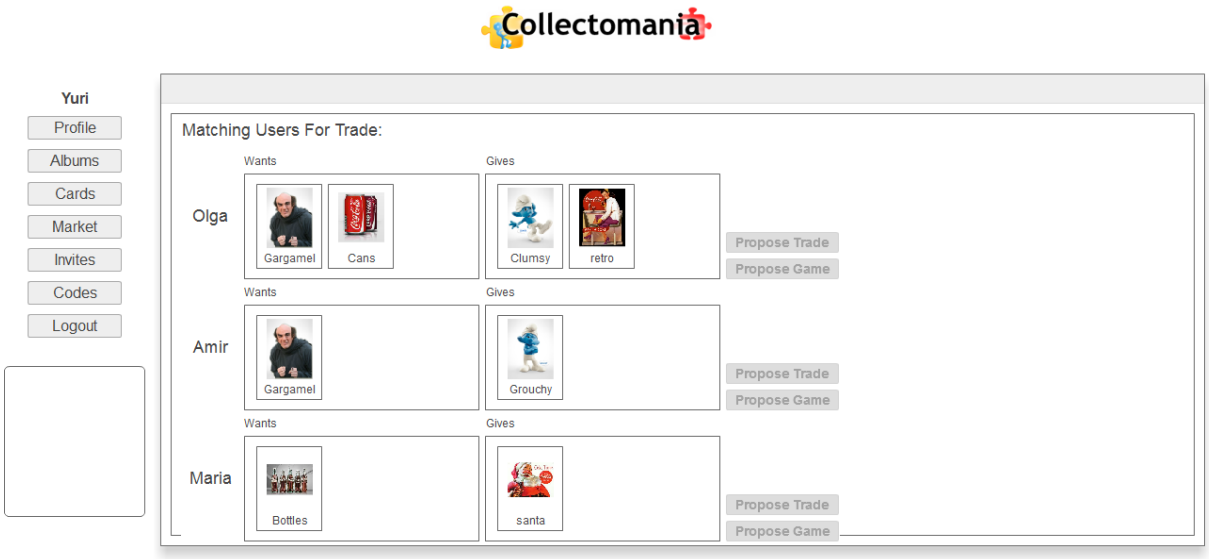
This window is reached by clicking in the "Cards" button in the menu.

On this window the user can see 3 lists:

1. Wish-list (was described in the previous section)
2. Trade-list – the list of cards the user doesn't need and he wants to trade them or play on them (more details about this subjects in the sections 7 and 8)
3. The list of all cards the user has – the user can select one of the shown cards and add it to his trade-list by clicking on the "Add" button. The user can't trade pasted cards, so it is not allowed to add pasted cards to trade list. Pasted cards are very bright, and can be recognized in this manner (like Papasmurf).

The user can also remove cards from wish-list/trade-list by selecting one of the cards and clicking the "Remove" button.

1.7 *Market Window*



This window can be reached by clicking on the “Market” button in the menu.

On this window the user can see users he can trade cards or play on cards with them. The system finds these users automatically by maximum compatibility with the current user.

The user selects the card he is ready to offer the other user and the card he wants from the other user. Then he selects one of two options: Trade or Game by clicking appropriate button.

When the request is sent, the sending user gets an appropriate notification. If the user has selected the Game option, the game is stopped and he moves to a waiting mode, like this:

The receiving user gets an invitation for trade or game and it is possible to indicate by the appearing number of invites on “Invites” button, like this:

1.8 Invites Window

This window can be reached by clicking on the “Invites” button in the menu.

On this page the user can see all the requests, he has got from other users.

The user can chose to confirm or decline. If the request is a simple Trade, and he confirms the request, then the users switch cards. The sending user receives notification about the performed trade. If the request is a Game, he goes to “Mini-Game”.

If the user declines the request nothing happens, except the sending user receives notification about the declined request.

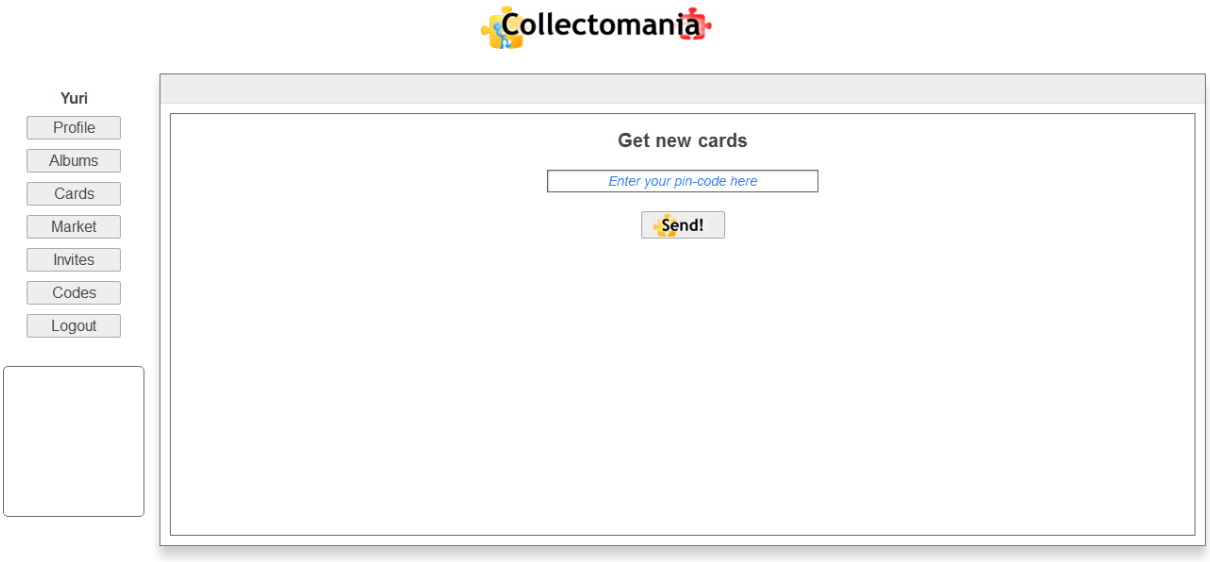
1.9 Mini-Game

Each user in turn makes his move. When his opponent makes his turn, he can see the description of turns below the pictures of moves. For example:

After each turn the users get notification about the winner and the loser of the turn. In the end of the game, they receive notification about the winner and loser of the entire game,

The winner receives his opponent's card and keeps his.

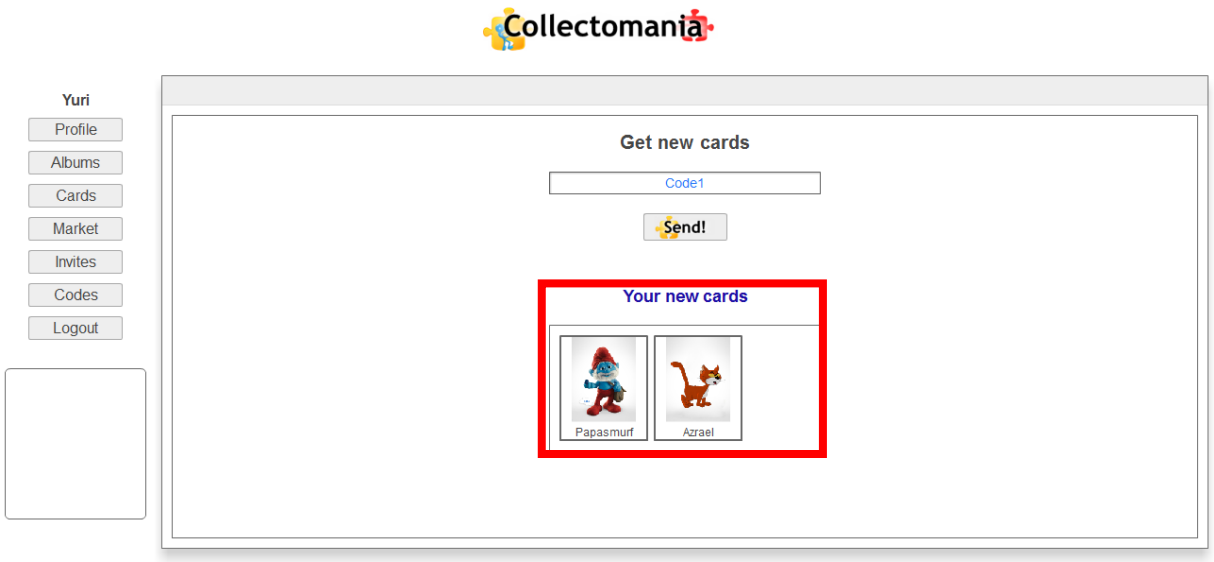
1.10 Codes Window



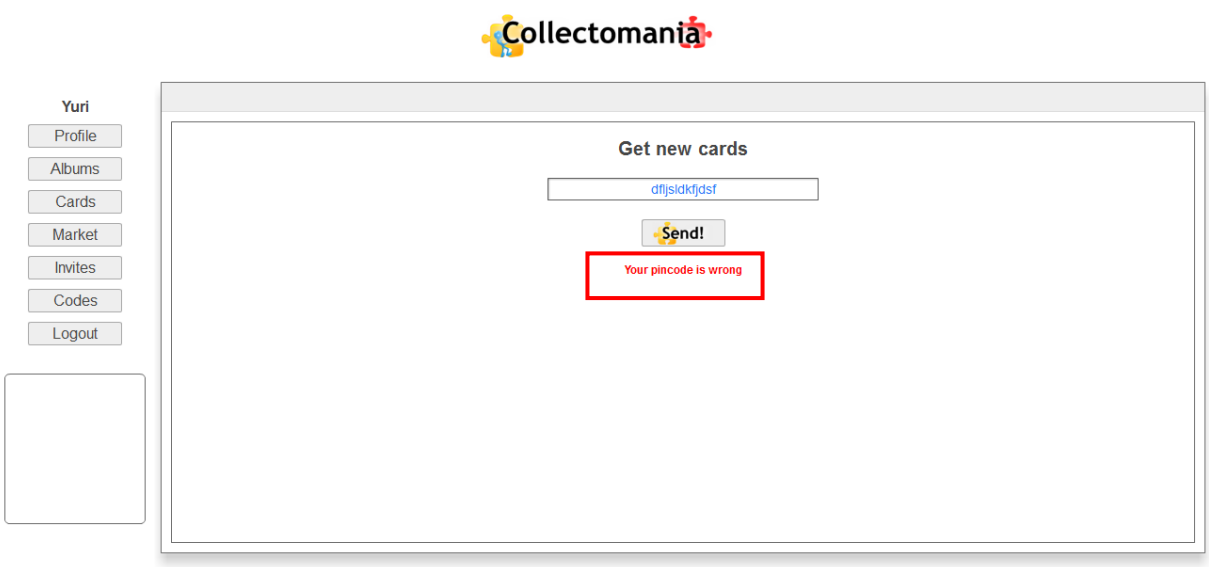
This window can be reached by clicking on the “Codes” button in the menu.

Each user can receive a secret code for a card (or even some cards). The way, the user can get his code is defined by the owner of the album.

The user can enter this code in an appropriate place on this window and sends the code to the system by clicking “Send” button. If the code is valid, the user can see below his new cards, like in this picture:



If the code is not valid, the user receives an error message:



2 Install and run

2.1 Technical Requirements

- Eclipse IDE for Java EE developers. Helios version will be enough.
- Tomcat 7
- BlazeDS.war file
- Subclipse – plugin providing support for Subversion within the Eclipse IDE.
- Adobe Flash Builder v4.5x
- SQL Server – MySQL v5.5.22
- MySQL Workbench – MySQL Workbench gpl v5.2.38
- Debug Flash player for your current version of Firefox browser (not mandatory)

The newest versions of tools should also work fine, due to back support and compatibility.

2.2 Installation

1. Download and install "Eclipse IDE for Java EE developers" from official website http://www.eclipse.org.

2. Next download Tomcat server from website http://tomcat.apache.org/download-70.cgi install it and integrate with eclipse. You can also install Tomcat plug-in for Eclipse, but it is not mandatory. Do not change the default settings during the installation. Pay attention that the running port should be set to 8080.

3. Download BlazeDS package from https://www.adobe.com/cfusion/entitlement/index.cfm?e=lc_blazeds. Then extract files. In fact you need only blazeds.war file. Put this file in Tomcat directory. This directory already includes directories: "bin", "conf", "lib" etc.

4. Download Subclipse. You can do it from within Eclipse: Help → Eclipse Marketplace… → search for Subclipse…

5. Install Adobe Flash Builder version 4.5. Once you completed the installation, you can now install the plugin for Eclipse.
    5.1. Navigate to the installed Flash Builder installation location and open the utilities folder.
    5.2. Run the executable Adobe Flash Builder 4.5 Plug-in Utility.exe.
    5.3. Select the language and click on OK
    5.4. Select the Flash Builder installation location if prompted.
    5.5. Select the Eclipse folder into which you want Flash Builder 4.5 to be plugged into and click 'Next'. (Note: Your copy of Eclipse must be version 3.6.1 or later, 32-bit and must contain a folder named "dropins")
    5.6. Review the pre-Installation summary and click on Install
    5.7. Following installation it is recommended that you edit the eclipse.ini file for your Eclipse instance, so that it includes the following settings:

    -vmargs -Xms256m -Xmx512m -XX:MaxPermSize=256m -XX:PermSize=64m

6. Download MySQL Server from http://dev.mysql.com/downloads/mysql. Download MySQL Workbench from http://dev.mysql.com/downloads/workbench. Install. Start from MySQL Server. During the installation do not change the default settings, especially the port. Its number should be 3306. Set username with "manager" and password with "1234".

7. In order to install debug version of Flash player, remove first the current version of Flash player. As was written before, debug version is not mandatory, but it is helpful during the debug of Flex code.

2.3 Download the Code and Run

1. Create new workspace for this project. If you use existing workspace, which was created before Tomcat was configured, you may have some strange errors.

2. Download the project from repository.

Open Eclipse → File → New → Other → SVN → Checkout projects from SVN → Click 'Next'.
Then, choose option 'create new repository location' and click 'Next'.
Type the following repository address:
https://collectomania.googlecode.com/svn/trunk and click 'Next' once again.
Expand the repository location and select folder 'collectomania-prototoype'. Click 'Next'.
Select option 'Check out as a project in a workspace'. You may change the project name from 'Flex-Java' to any other, but in further explanation the name 'Flex-Java' will be used. Do not change any other default options in this wizard. Just click 'Next' and 'Finish'.

At this point, you may have errors in your project, but should not have red lines in code. You need to make changes in 2 configuration files.

3. In workspace folder open (with notepad) file '.actionScriptProperties'. In line 3 you can find the path to workspace and the project. Change it to path on your PC.
In workspace folder open (with notepad) file '.flexProperties'. In line 2 you can find the path to blazeDS.war. Change it to path on your PC.
Save changes.

Now you have all the code on your PC, but the database is empty.

4. Open MySQL Workbench and create new server connection. As was mentioned above, username should be set to 'manager' and password to '1234'. Create new SQL schema. The name of the schema should be 'collectomania1.0'. Now navigate in downloaded project to Flex-Java → src → dbaccessors → collectomania1.0.sql
Run this script in order to init the database.

5. Now you can run the project. Navigate in Eclipse to Flex-Java → flex_src → Collectomania.mxml
   Right click on the file → Run as → Run on server.

By default, eclipse opens its own browser and you can see the code for this page. It is not recommended to develop the project with this browser. Use Firefox instead, especially if you installed the debug version of Flash player.

6. Open your browser and type http://localhost:8080/Flex-Java/Collectomania.swf.

2.4 Important Remarks for Further Management

- When you commit a new version to SVN, please make sure that you do not commit your versions of '.actionScriptProperties' and '.flexProperties'. Otherwise, your coworkers may have some nasty errors.

- If you want to init the database, drop all the tables in schema, but do not drop the schema and run the script from 'collectomania1.0.sql' file.

- There is a website with all documentation http://collectomania.tripod.com. If you want to continue manage this site, you should login to http://www.tripod.lycos.com with
  Username: collectomania
  Password: collect4

3. Developer Manual

3.1 High Level Architecture Design

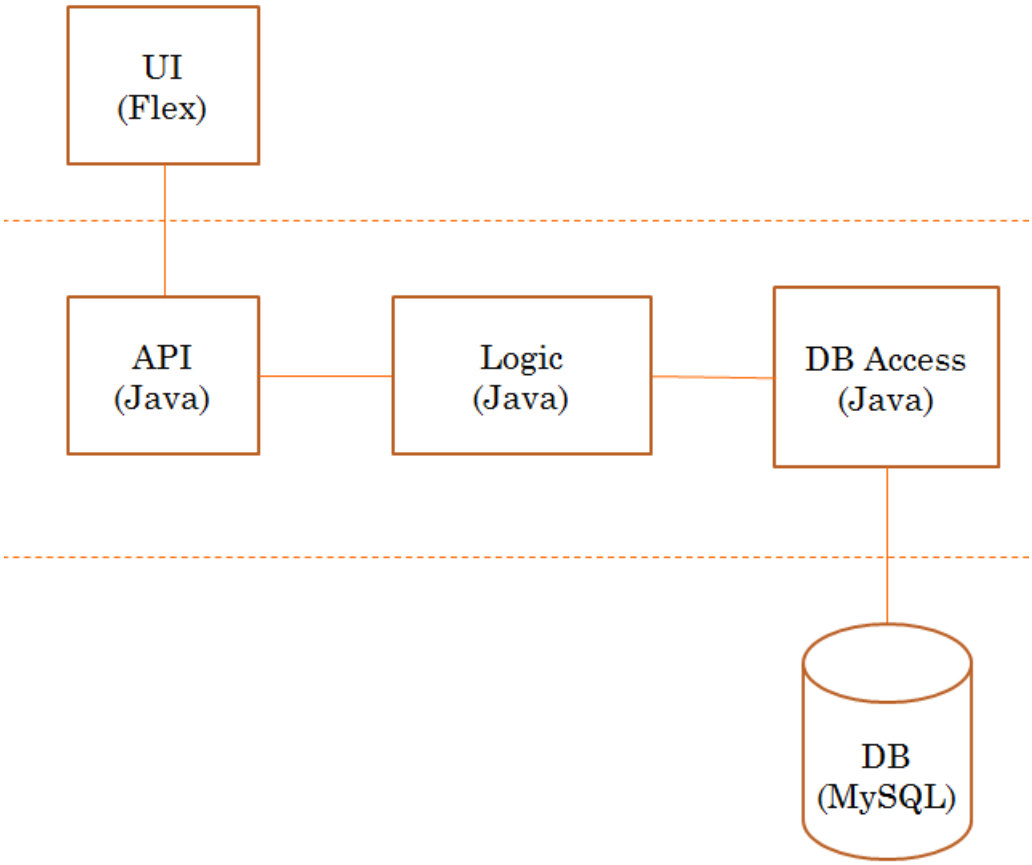
3.1.1 Design:

The system is compound of three main parts:

-The User Interface (Written in Flex)

-The Server Logic (Written in Java)

-The Data Base (MySQL)

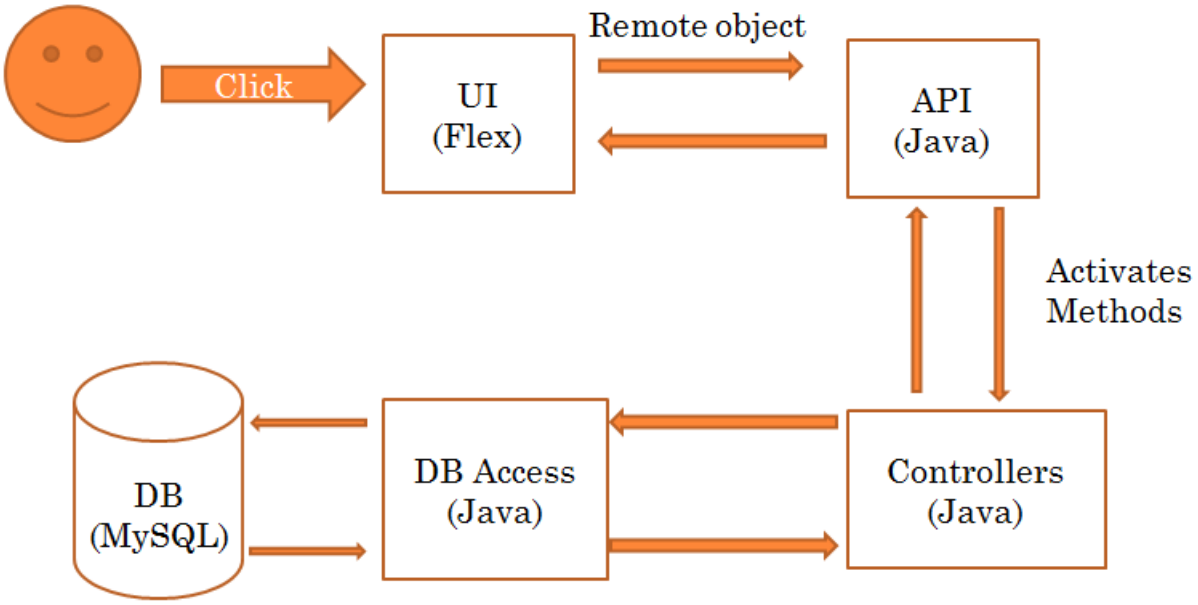


Each part will be described in its own chapter.

3.1.2 Data Flow:

The application works as most web based applications do. A user click generates an event at the client side which calls a specific function on the server, the server logic handle the request and generates the answer data (querying the database if needed). The data is packed in a 'data object' which then get serialized and sent back to the client.

The following diagram illustrates the procedure:



The next chapters will go deeper into each of these parts and describe the design & flow of data in a more specific way.

3.2 The User Interface

The user interface (UI) is a collection of Flex components. All resides under the 'flex_src' folder.

3.2.1 Main:

Collectomania.mxml is the main application page, it contains the **mainMenu** module and a ViewStack module. These two modules are working together to create the client side MVC. A view stack is a Flex container component that holds other components and display them as needed, the main menu is just a collection of buttons each calls this.changeMainPage(i) to change the displayed page.

3.2.2 Pages:

A page is a Flex container just like all other Flex component.

In collectomania pages resides in the main menu, only one page is showed at a time, this is the active page.

A page is loaded into memory the first time it is shown on screen, this is when the page **initialization** function is called automatically when the pages' build is complete. When navigating to pages that have already been initialized it is the developer responsibility to call the init function if needed

Example: When a user logs in the init function of the main page loads the user data (the exact method will be described later under client\server communication). This happens because the page has just been created. If the user now logs out and logs in back again the main page will show but will **not** be created again – thus the init function will not get called and the old user data will show.

To fix this, in the main menu component, every time a page is displayed it's being checked to see if it's the first time and if not the init function is called implicitly.

Most pages have the same layout for the code, starting with the Action Script code followed by the **mxml** component that constructs the page. Each component can be a simple component like a button or a collection of components. It can also be a custom component which is a template container.

3.2.3 Custom Components:

Custom components are all in the '/components' directory of the project. They are actually very similar to pages in a way that they are components that can be nested in other containers. The only different is semantically. We decided to call a page to every component which we display in the main viewStack.

3.2.4 Action Script Code

Action script to MXML pages is what JavaScript is to html, it moves objects around and updates the viewable components on screen. There are two kinds of changes that are being done with code:

Local visual changes

Some code just handles visual animation, changing the active screen, or validating user input before it is being sent to the server.

Remote Procedure Calls

Some code is responsible for communication with the server; the main page Collectomaina.mxml defines an object of type 'RemoteObject' with name 'uiController'. This is a **Blaze Remote Object** described in '/flex_src/WebContent/flex/remoting-config.xml'. There it is binded to the java API class.

It is now possible to call each method the uiController reveals in java using Action Script. The java method will run returning the answer in a callback defined under the main page, where uiController is defined.

The call itself is being done by writing something like _API().methodName(Parameters);

Example: In the welcome screen the initData() function calls _API().getActiveAlbums(); . **_API** is a global objects which returns the uiController from anywhere in the AS code. Do note that include "../scripts/scripts.as"; must be written in the top of the page script in order to user this call.

The callback for getActiveAlbums is 'welcomeScreen.albumResultHandler(event)' as can be seen in Collectomaina.mxml on line 31. 'albumResultHandler' Will check that the answer is valid , parse it to AS objects, update the page data to reflect the new albums it just got and finally display then on screen.

Communication between Users

In addition to user-server communication type, there is also user-user communication needed. This mechanism is used when users want to trade cards or play with each other. In fact, communication between users is passes through the server. The difference from **Remote Procedure Calles** described above, that the client side does not wait for a reply (callback). The client receives messages from other users asynchronously. (One good tutorial is found here http://www.flex-tutorial.fr/wp-content/uploads/2009/01/data-push-in-flex-using-blaze-ds.pdf)

This piece of functionality is also implemented with **BlazeDS**. BlazeDS provides additional remote object called **Consumer**. Its job is to listen to the server and receive the messages

that were directed to it. Each consumer belongs to channel via adapter. The configurations are located in '/flex_src/WebContent/flex/messaging-config.xml'. There are two consumers for each client: **InvitesConsumer** for receiving invites and notifications from other users and **rpcConsumer** which established when RPS game started. Each consumer has its own message handler function. Once message is received, the handler function is called. Handler is described in "message" attribute of declaration.

Take a look at **InvitesConsumer** in **Collectomania.mxml**. It is declared right after remote object declaration. Consumer object won't receive any message until it has been subscribed to the server. Right before the subscription is needed to specify the message selector. Selector filters the messages, and reacts only to messages that are fit to selection criteria. In fact, the selector is just a header in received message. As every header, it has its name and property.

Example: InvitesConsumer is declared in Collectomania.mxml, but the subscribtion is performed only when user complete the login procedure. In loginHandler in LoginPage.mxml the last line of the handler function calls to InviteConsumerSubscribe(Username.text).
In this short method we want to receive messages for logged in user only. Therefore, we specify the selector header "toUser" to be a username of logged in user. Remember, that the invariant of the system is that each user has unique username.

```
invitesConsumer.selector = "toUser ='"+username+"'";
```

Each consumer object may have more than one selector, but the developer has to care for the appropriate selector when message is generated on server. Note that the selector definition must have SQL condition format.

(There is also additional way to perform the message selection using subtopics, but it is not convenient way of implementation of collectomania needs.)

There are additional notes in server side. Package 'asyncsenders' contains classes that generate the appropriate messages and send them to consumers. Each such class must extend **ServiceAdapter** class of BlazeDS architecture and override the method invoke(Message message). Take a look at Invites sender. Most of its methods are generates the messages with suitable headers and selectors. At the end of each method there is a call to routeMessageToService(msg,null). Behind the scene it performs the filtering and then overridden method invoke is invoked. **UserController** object in server has its own **InvitesSender**. The way of message generation and sending is easy for tracking in Eclipse.

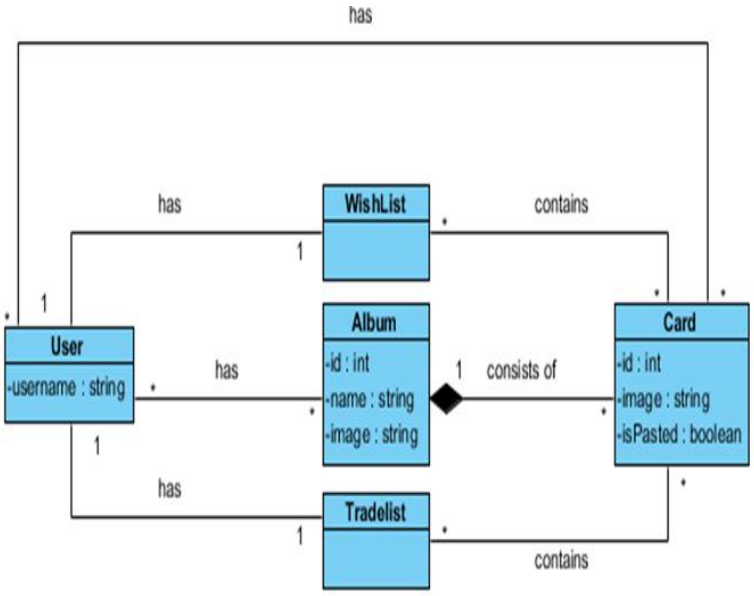
3.3 The Server API

As you have seen in the previous chapter, the Flex application can call every method revealed by the server by using the uiController object. The java source files are in '/src/API' folder. The interface API.java contains the following methods:

- register(String, String, String, String) : boolean
- login(String, String) : boolean
- signout(String) : boolean
- enterPincode(String, String) : ArrayList<CardInstance>
- getNotifications(String) : String
- getUserDetails(String) : User
- getActiveAlbums() : ArrayList<AlbumTemplate>
- pasteCard(String, int, int) : ArrayList<Integer>
- findMatchingUsers(String) : ArrayList<User>
- createTradeInvite(String, String, int, int) : boolean
- createGameInvite(String, String, int, int, String) : Invite
- getInvites(String) : ArrayList<Invite>
- confirmInvite(String, String, int, int, String) : Invite
- declineInvite(String, String, int, int, String) : boolean
- cancelGameInvite(String, String, int, int, String) : boolean
- makeTurnRPC(String, String, String) : String

We will not go over each one of those as they are clearly described in other documents. The main think to understand is that every method here can be called from the Flex application and will return a data object in java. It then will be parsed to an AS object and handled by the UI.

Data Objects looks similar to this one:



Note:
The server will build this objects from the data base, send them (serialized) to the client using blazeDS and destroy them immediately, the server *never* keeps data objects in memory beyond the purpose of sending them to the client.

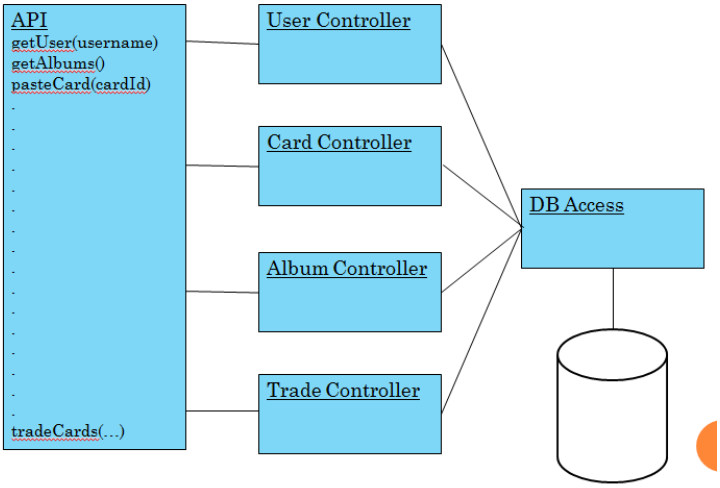
3.4 The Server Logic

3.4.1 Controllers:

The API implementation (UIController.java) will dispatch the call from the client to one of the following Controllers:

- AlbumController
- CardController
- GameController
- PinCodeController
- UserController

The following diagram illustrates this:



Each such controller handles the logic it needs to return an answer. All the code in the controllers classes is pure Java, commented, documented and mostly self-explained.
When a controller needs to get information about the **state** of the system (i.e. is player x online? What cards he have? etc..) it uses the DB Access class to (surprisingly) access the database.

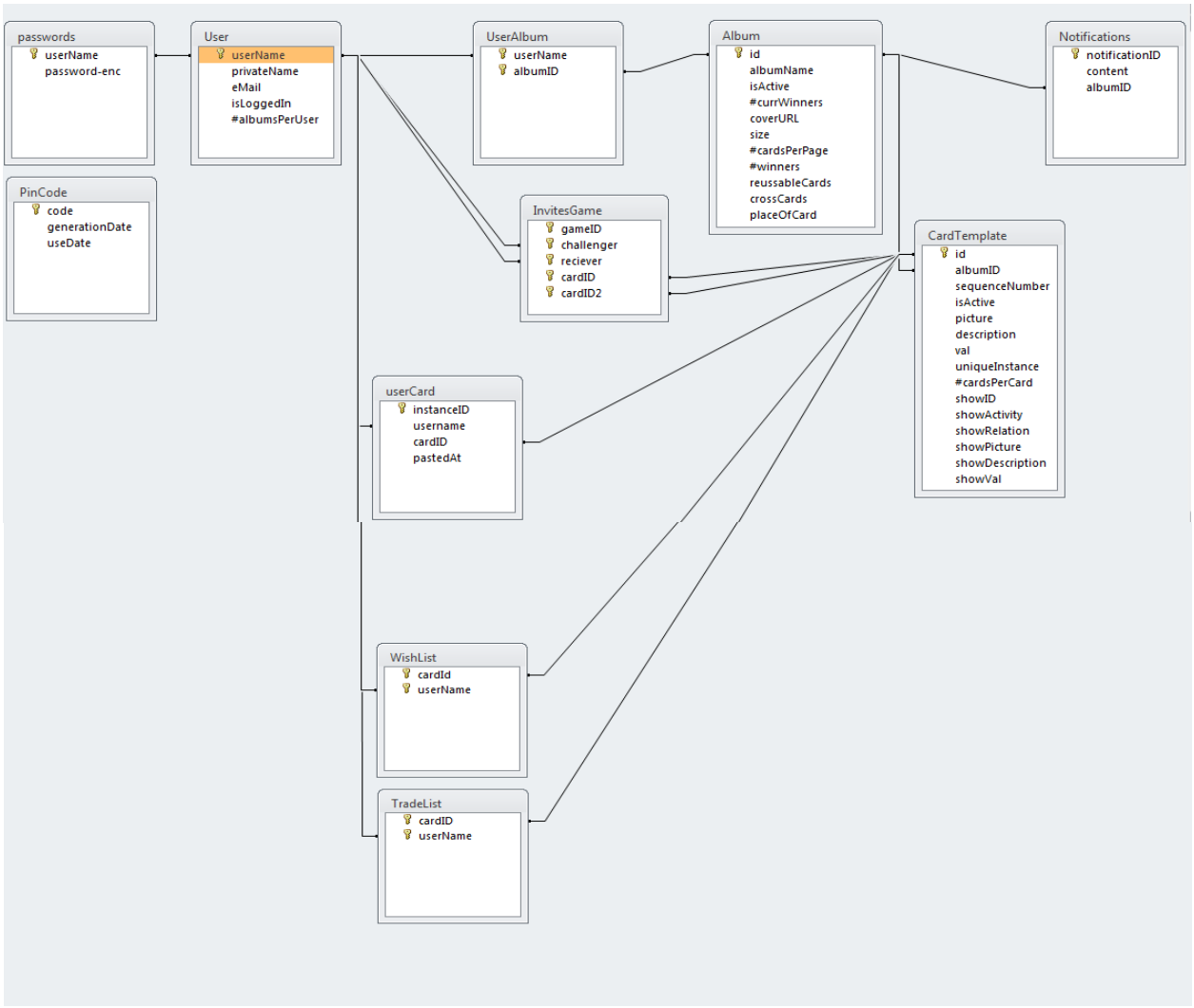
3.4.2 DB Access:

This class holds a connection to the MySQL Database, it is implemented as a singleton so the connection is always alive (In opposed to connecting on each request). It is important to know that this is the only class that interact with the database and designed to remain that way. Each Controller must keep a reference to it and call it upon demand.

3.5 The Data Base

As mentioned before the server is **stateless** (no java data is being kept in memory, only the logic) thus all the state of the system (like users, cards, sessions and codes) are being kept in the DB.

The tables and their connections are described in the following diagram:



4. Testing

4.1 Unit testing:

First of all, we perform tests from ADD.

Method: FindUserByWishList(uid)							
CaseID	**Goal**	**Input** **uid**	**Pre-conditions**	**Post-conditions**	**Expected Output**	**Observed Output**	**Test Pass**
1	At least one user has one of wished cards	test@mail.com	User test@mail.com has at least one card in his wishlist and there is at least one suitable user	None	List of users (except of test@mail.com) grouped by card	List of users except of test@mail.com grouped by card	V
2	No user has wished cards	test@mail.com	User test@mail.com has at least one card in his wishlist but there is no suitable user	None	Empty list of users	Empty list of users and message “There are no invites for you”	V
3	User has no cards in this wishlist	test@mail.com	User test@mail.com has no cards in his wishlist	None	Empty list of users	Empty list of matched users in Market tab.	V

Method: trade(uid1, uid2, cid1, cid2)									
CaseID	**Goal**	**Input**				**Pre-conditions**	**Post-conditions**	**Expected Output**	**Observed Output**	**Test Pass**
		uid1	**uid2**	**cid1**	**cid2**					
1	Legal trade	test1@mail.com	test2@mail.com	111	222	User test1 has card 111, user test2 has card 222	User test1 has card 222, user test2 has card 111	Both users receive message about successful trade	Both users receive message "Trade with <username> on cards <cardname1> and <cardname2> confirmed"	V
2	One of users does not have appropriate card. (Card might be lost in game or traded)	test1@mail.com	test2@mail.com	111	222	User test1 has card 111, user test2 does not have card 222	None	User that asked to complete the trade receives an error message.	User that asked to complete the trade received message "trade cannot be completed"	V
3	Both users have no cards for trade	test1@mail.com	test2@mail.com	111	222	User test1 does not have card 111, user test2 does not have card 222	None	User that asked to complete the trade receive an error message.	User that asked to complete the trade received message "trade cannot be completed"	V

Method: EnterPinCode(code, uid)								
CaseID	**Goal**	**Input**		**Pre-conditions**	**Post-conditions**	**Expected Output**	**Observed Output**	**Test Pass**
		code	**uid**					
1	Valid not used pincode	123456789	test@mail.com	code 123456789 is legal and was not used before	Cards that belong to this code added to user's card list	Success message	New cards was shown to the user	V
2	Valid used pincode	123456789	test@mail.com	code 123456789 is legal code, but was already used	None	Error message "Illegal pincode"	Message “Your pincode is wrong” received	V
3	invalid pincode	123454321	test@mail.com	Code 123454321 is illegal	None	Error message "Illegal pincode"	Message “Your pincode is wrong” received	V

In ADD we specified only the tests for unique functionality of Collectomania system. Of course we also checked classical illegal user inputs like empty string, very long string etc. The only places were user can insert strings is ‘login’ window, ‘registration’ window and ‘pincode’ window. In all of the fields the case of empty string is handled in action script level. Therefore, there is no additional overhead on the server. Maximum string size is specified in static XML code. The attribute `maxChars` of `TextInput` node sets the maximum string length in field. Again, it avoids from additional call to the server.

In addition we added more test cases to the test suite and run them. All units of the server were tested apart according to the logic of the requirements.

Major tests:

	Name	Purpose	details	Unit	Legal input result	Illegal input result
1	Communication test	Test the communication between server and client	Messages were sent from the client and checked at the server, and vice versa	API	The communication worked. All messages were received at the destination	
2	Paste card	Test the “paste card” use case	User triggered the paste card scenario. Check the card was pasted	Card controller	The card was pasted.	Illegal actions were announced to user.
3	Asynchronous response	Check the “push messages” mechanism. Make sure all messages are received	When trading cards, the offering user received notification when the other user accepted the offer	Asyncsenders	User had received the notifications	
4	Trade card	Check the trading use case. Make sure cards “change hands”	When a trade is made, the cards should switch owners	Card controller	Cards switched owners	Invalid input was announced to user.
5	Enter pin code	Check the pin code mechanism. Make sure only relevant cards received.	When inserting a legal pin code, the user will get the cards linked to it.	Pin code controller	User received the relevant cards. Pin code marked a used	User receives a relevant message
6	Data base tests	Check legal queries are performed and no SQL injection allowed	Check legal queries are performed and no SQL injection allowed	Data access	Sql queries were correct	User can’t insert sql injection sentence.

4.2 Integration testing:

All units were tested combined to make sure they coordinate and remains encapsulated (without “high coupling”).

At this point we repeated all tests that were performed in the unit testing phase, this time combining several controllers including the actual data base by using the data access unit.

The main purpose of this part was to make sure all use cases earlier defined were implemented correctly.

Major tests:

	Name	Purpose	Details	Unit	Legal input result	Illegal input result
1	Paste card	Test the “paste card” use case.	User triggered the paste card scenario. Check the card was pasted	Card controller, User controller, UI controller, Data access	The card was pasted.	Illegal actions were announced to user.
3	Asynchronous response	Check the “push messages” mechanism. Make sure all messages are received	When trading cards, the offering user received notification when the other user accepted the offer	Asyncsenders, UI controller, Data access	User had received the notifications	
4	Trade card	Check the trading use case. Make sure cards “change hands”	When a trade is made, the cards should switch owners	Card controller, User controller, UI controller, Data access	Cards switched owners	Invalid input was announced to user.
5	Enter pin code	Check the pin code mechanism. Make sure only relevant cards received.	When inserting a legal pin code, the user will get the cards linked to it.	Pin code controller, UI controller, Data access	User received the relevant cards. Pin code marked a used	User receives a relevant message
6	Play game	Check the game mechanism. Make sure the winner gets the lot	When a game is played, the winner gets all cards. If tie, no switch made. Async. Messages should pass	Game controller, Asyncsenders, Card controller, UI controller, Data access	Messages passes through, Winner get cards, Looser lose cards, If tie – no switch is made	Relevant message appears to relevant user.

4.3 Scalability

Scalability tests were not performed since the project is a prototype and a POC.

4.4 Availability

We performed availability tests by running the application on several computers, while collecting failures and exceptions.

We logged with several users simultaneously in order to check the multi-user support of the server.

Results: no failures occurred, no exceptions were thrown.

4.5 Compatibility:

a. We tested the application on several web browsers (IE, Firefox and Chrom), several hardware platforms (Dell, HP, IBM). All tests were successful.
b. We tested the application on several devices: Laptop, PC, smartphone (Samsung – Android). Since the user interface is written in FLEX, the application cannot be ran on Apple devices.
c.

4.6 Security:

SQL Injections:

In order to avoid SQL Injections, we used parameterized queries mechanism implemented in Java as PreparedStatement, thus not allowing harmful input into the system. Here is an example:

```
Statement = this.conn.prepareStatement("INSERT INTO
`collectomania1.0`.`tblpassword` VALUES (?,?)");
statement.setString(1, username);
statement.setString(2, password);
```

In addition, we minimized as possible the user free input and used mostly build-in content.

4.7 Stability:

We checked the response of the system to several common errors:

d. Illegal input – we checked what happens if the user inserts illegal input in every screen the user has access to.

e. Wrong input – we checked what happens when user selects something impossible (e.g. trade a card he doesn't have)

To avoid the exceptions being thrown on the server side, we used the "try and catch" mechanism.

```
try {
            PreparedStatement statement =
            this.conn.prepareStatement("INSERT INTO
            `collectomania1.0`.`tbluser` VALUES (?,?,?,0,NULL)");
            statement.setString(1, username);
            statement.setString(2, name);
            statement.setString(3, email);
            updateQuery1 = statement.executeUpdate();
    } catch (SQLException e) {
            e.printStackTrace();
            return false;
    }
```

Flex treats every addressing to null object as an exception, even if it's a read only statement of the form – if(obj == null)

So, to avoid exceptions on the client side, we made sure we don't pass any null objects to the client. Instead, we chose to send some pre-defined values that will indicate that an illegal action\exception accrued. The result is analyzed on the client side and, when necessary, an error message is shown to the user using the message panel on the bottom left corner of the screen.

Results: the system handles all exceptions and illegal inputs.

4.8 Localization and International:

We checked that the application can be ran on devices with different regional setting: Latin(USA, UK …) Arab (Egypt, Jordan…) Far East(China, Japan…) Cyrillic (Russia, Ukraine…) and as long as the web browser language is set to English, the application runs flawlessly.

In other cases, it depends on the browser settings.

5 Future extensions

The future possible extensions of the code are related to the adding of the “Manager” to the system.

As it has been described in the ARD document the idea behind the project is to give commercial companies a new possibility to advertise themselves. So each company can open an album and assign a manager who is responsible for the album. This entity has to be registered as the manager to the system (with permission of the system administrator of course).

When a user introduces him as a manger, the system has to check it, and if it is a manager a different window is opened for the manager. He can see his published albums, unpublished albums and of course to open a new album.

When the manager opens a new album, he decides about the number of cards on every page and the number of pages. He loads selected files to his album (pictures, video, etc). He has an ability to save his album on every part of the loading: the unfinished album will be saved as “unpublished”, and wouldn’t been seen to the simple users (who collect albums), until the manager decides to publish it.

After loading the necessary files to the album, the manager chooses values for possible rules, for example: the number of possible winners’ the number of possible users of the album, the maximal number of cards to play on, etc.

When the manager finishes creating the album and publishes it, the album appears in the list of available albums, and the option of start a new album should be added to the system. It means that the user can click on this album and start collecting its cards. Then the album appears in his list of albums.

It is necessary to pay attention that before the implementation itself a few important things are need to be added:

1. Tables which describe Manager, Manager-Album (relationship between managers and albums) and tables of the possible rules.
2. Appropriate data objects in java (server side) and in flex (client side).

In addition to the above extensions, there is need to take care of some non-functional requirements, those were mentioned in the ADD and were not implemented in the presented project because of the changed requirements and the highlights from the customer’s side.