# Facebook Analyst app

Discover who your friends are common friends

Chai Botta 817333
Antonello Fodde 817371

# Table of Contents

# Preface

This application aims to get a practical experience of the presentation level web application development. In the project the Spring Framework will be used because nowadays it is a well know framework for web development. We will follow in particular the MCV approach. Also will be faced the problem of managing persistency, in particular the object-relational mapping problem and for this purpose the application will use one of the most successful approach by adopting Hibernate libraries. The entire web application will communicate with an external platform which is Facebook.com, by using its Graph API.

# Overview

## What the system is

The Facebook Analyst app is a web application which allows all the Facebook's users to discover relationship between their friends. The system let the user to connect his Facebook account with the web application and show some personal useful information related to the user such as name, nickname, location and personal image. It's shown also the number of friends. If the user click on the Friends button on the top-navigation bar the entire list of friends is shown. From the list it is possible to select all or just some friends and by this way a friendship graphical network is shown. The user can still discover more on his friends just by clicking on each node into the graph. Information such as networks metrics (Degree, Centrality and Betweenness) and friends' public data will be shown in a box.

## Who will use it

Even if the application comes out from a practical exercise, we develop it having in mind the possible end-users which are young-middle aged people (between 13-55 years old) used to technology who wants to discover more about their Facebook friendship, for example if some of their friends are also friends.

## Objectives

Our objectives in developing such a web application are:

1. Learn how the Spring Framework works
    a. Pro and Cons in using this framework
    b. Go deeper into the Dependency Injection and Inversion of Control Patterns
    c. How take advantage of the MVC approach developing web application
2. Learn how to setup a persistency layer
    a. By using a low-level approach with JDBC
    b. By using a high-level approach with Hibernate
3. Learn how to interact with external platform
    a. Facebook.com and Facebook Graph API
    b. Spring social library [only Facebook provider is configured]

4. Learn how to build a responsive and good looking web interface

## Out of scope and assumption

We assume any our end-users have a Facebook account and accept the Facebook term of use by allowing our application to access to their personal data and data related to their friends.

Third parties' logos, names and brands belong to their own company. We don't claim any property by naming them.

# Requirement Analysis

Before starting the application development we defined a set of Functional Requirements and Not Functional Requirements. They are listed below.

## Functional Requirements

The web application has to be implemented using Spring Framework and its must-have features are:

1. The end-user should be able to access the application using any browser
2. The user should be able to login to the application by using his Facebook account.
3. The application has to show a list of the user's friends
4. The user should be able to select some/all of his friends and the application has to show all the friendship relationship between the user and the selected friends. Only common friends between each selected friends are retrieved.
5. The application should be able to show a friendship graph with the list obtained at point 4.
6. The user should be able to double-click on each node of the graph and retrieve in this way more information related to that node, representing a friend, and the node importance according to the main network analysis algorithm.
7. The application has to compute for each node in the graph the following networks metrics:
    a. Closeness Centrality and the normalized one
    b. Betweenness Centrality and the normalized one
    c. Degree Centrality and the normalized one

## Not Functional Requirements

The non-functional requirements are documents, items and features not related to the objectives at the base of this project.

1. The project documentations
2. The application should have a good look (pretty interface)
3. The interface should be responsive according to the last best-practice for website development
4. Gamification and entertainment elements on HTML5 canvas
5. Performance issues

# Architecture and Design

## Architecture Definition

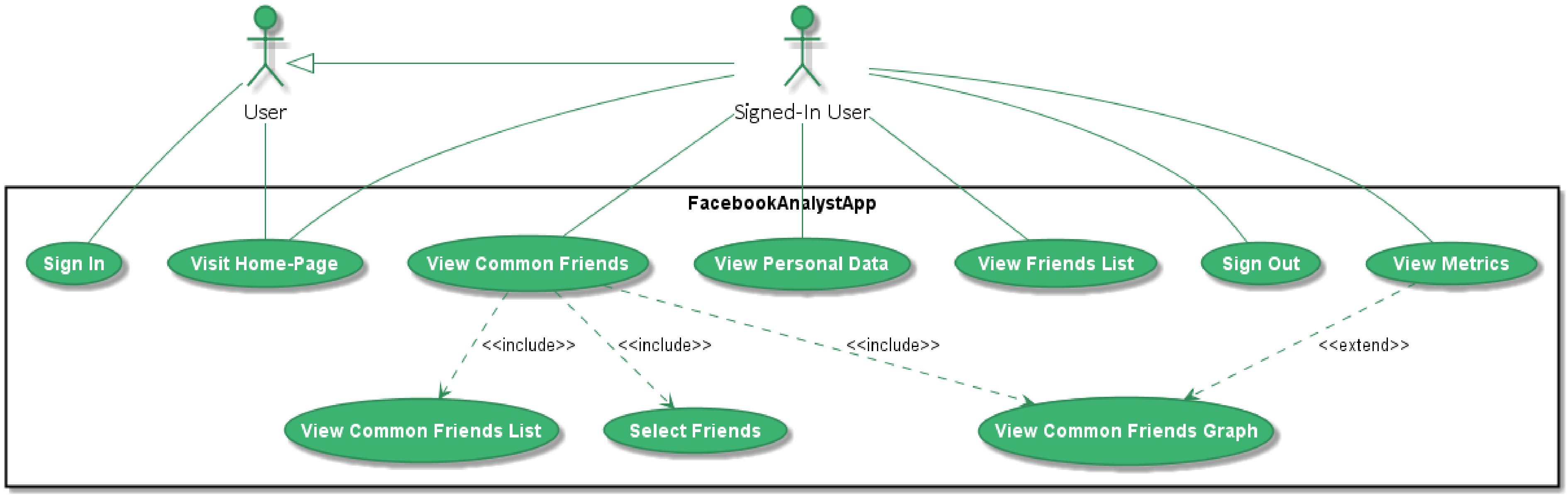
As most of the enterprise level web application, the Facebook Analysis web application is built on top of many different layers. The application's core is based on the server side which is developed in the MVC architecture. The front-end is instead developed using one of the most famous and used web framework for web- interfaces which is Foundation5.

The architecture and the software design were built on top the use cases we defined. In the next page we distinguished two users, one not yet logged and one logged into the web application. The diagram also shows how the users interact with the system (FacebookAnalystApp).

## Use Cases

We have basically two type of users:

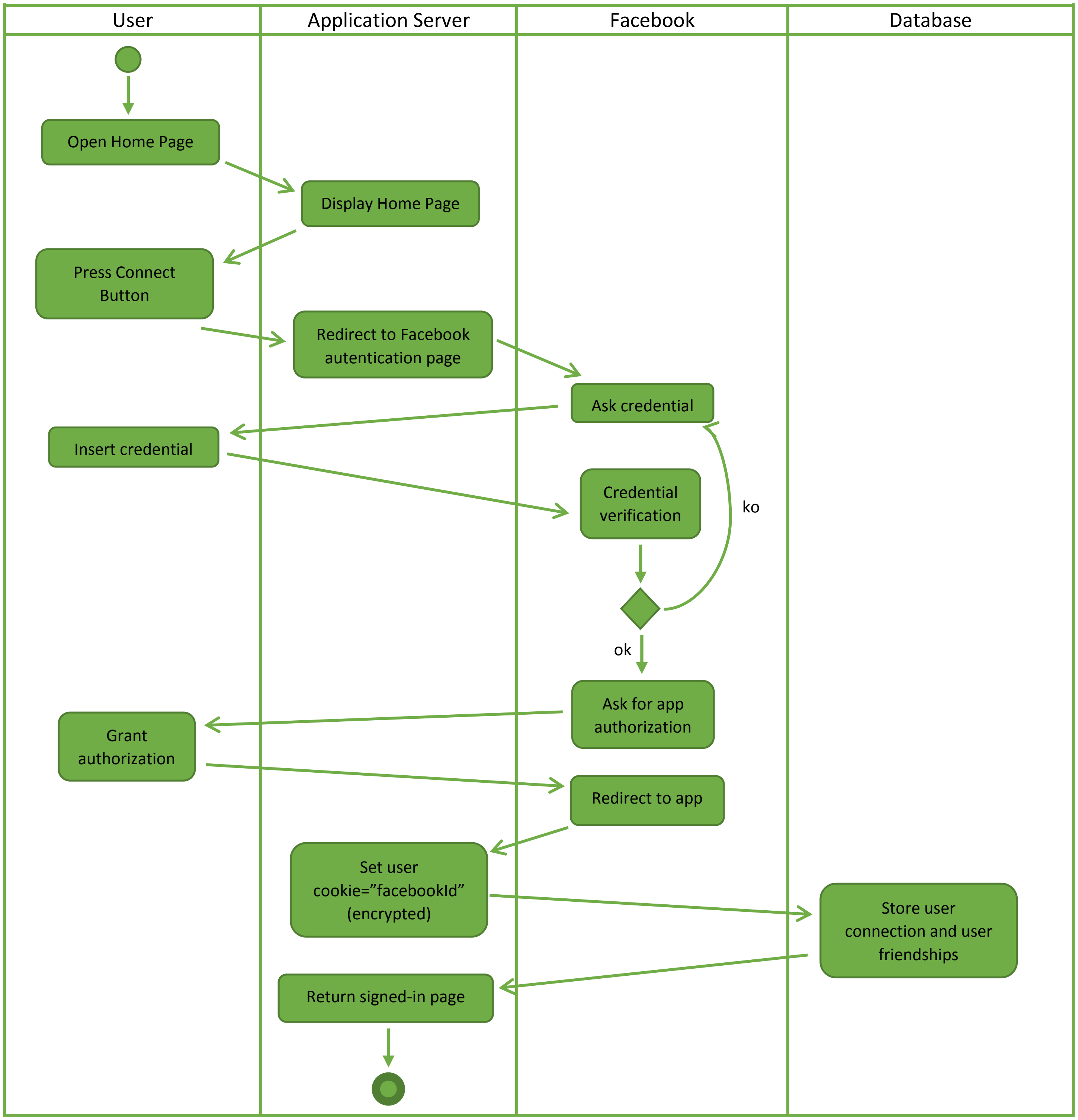
- **Not Signed-In User:** can only visit the home page and sign-in clicking on a button
- **Signed-In User:** basically can do all kind of operation specified by the requirements, and also perform a sign-out that ensures the deletion of all his personal data within the application.



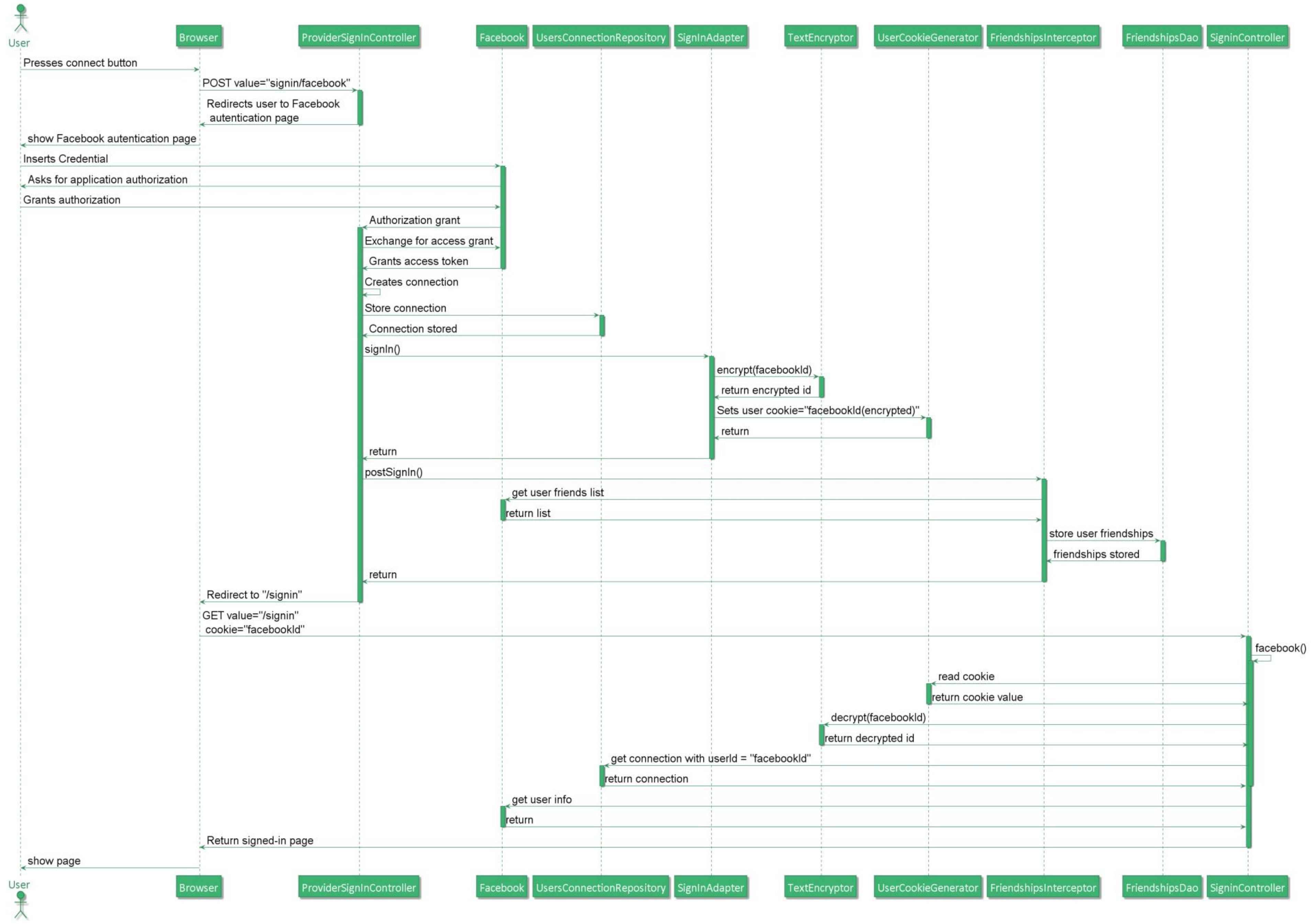
## Operations Flow

### Users Connection - Activity Diagram

After Home-Page visualization, a user can sign-in by clicking on the Connect Button. He will be redirected to the Facebook authentication page (skipped if he is already logged in and with Facebook login kept) when user can perform the access and then grant the required app authorizations. After that the application ensures to store the user's connection (access token) and all his friendships into the database. The application will use his Facebook id for identify him from now on and will set a user cookie with his id (that will be encrypted for security purposes) for recognize his following requests. We preferred to use a cookie to avoid a double log-in in order to use the app.

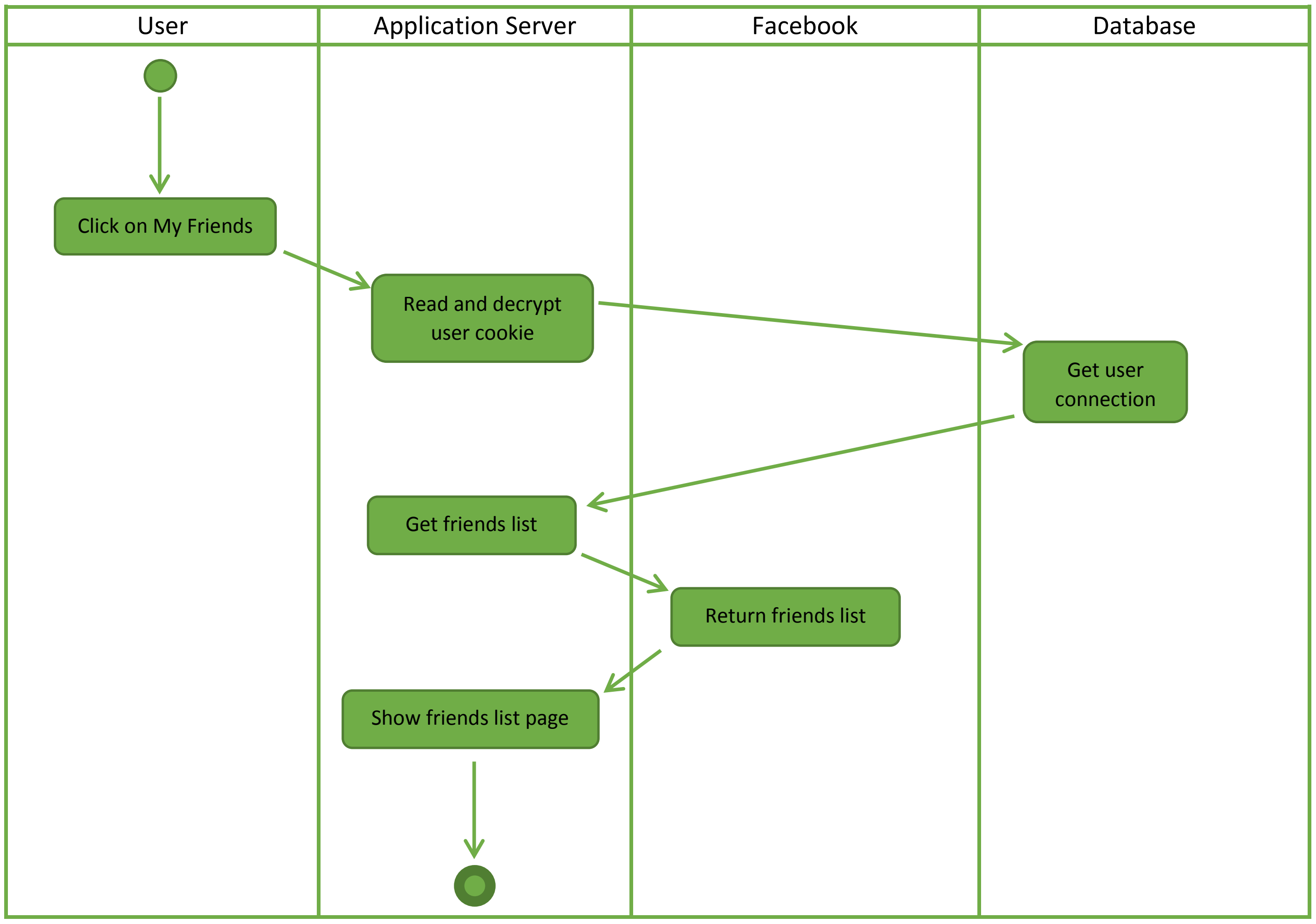


## User Connection in Detail - Sequence Diagram

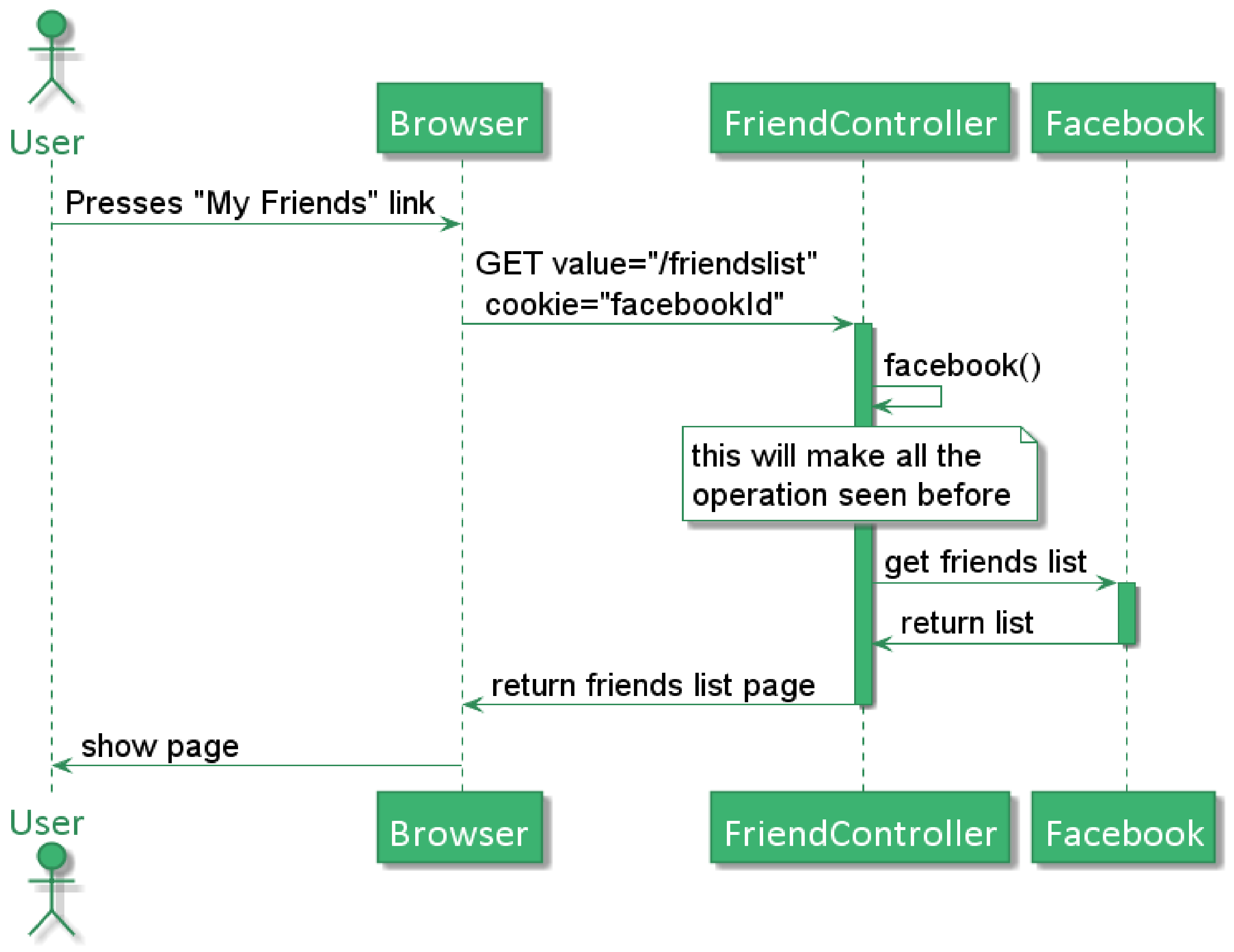


**Get Friends List - Activity Diagram**

After a user is correctly connected with his Facebook account, he can finally use the application and request his friends list. The application server when receives the request, it reads the cookie value after decrypting it and identify the user. Then get his connection form the database and after a Facebook API call it returns the user's friends list.

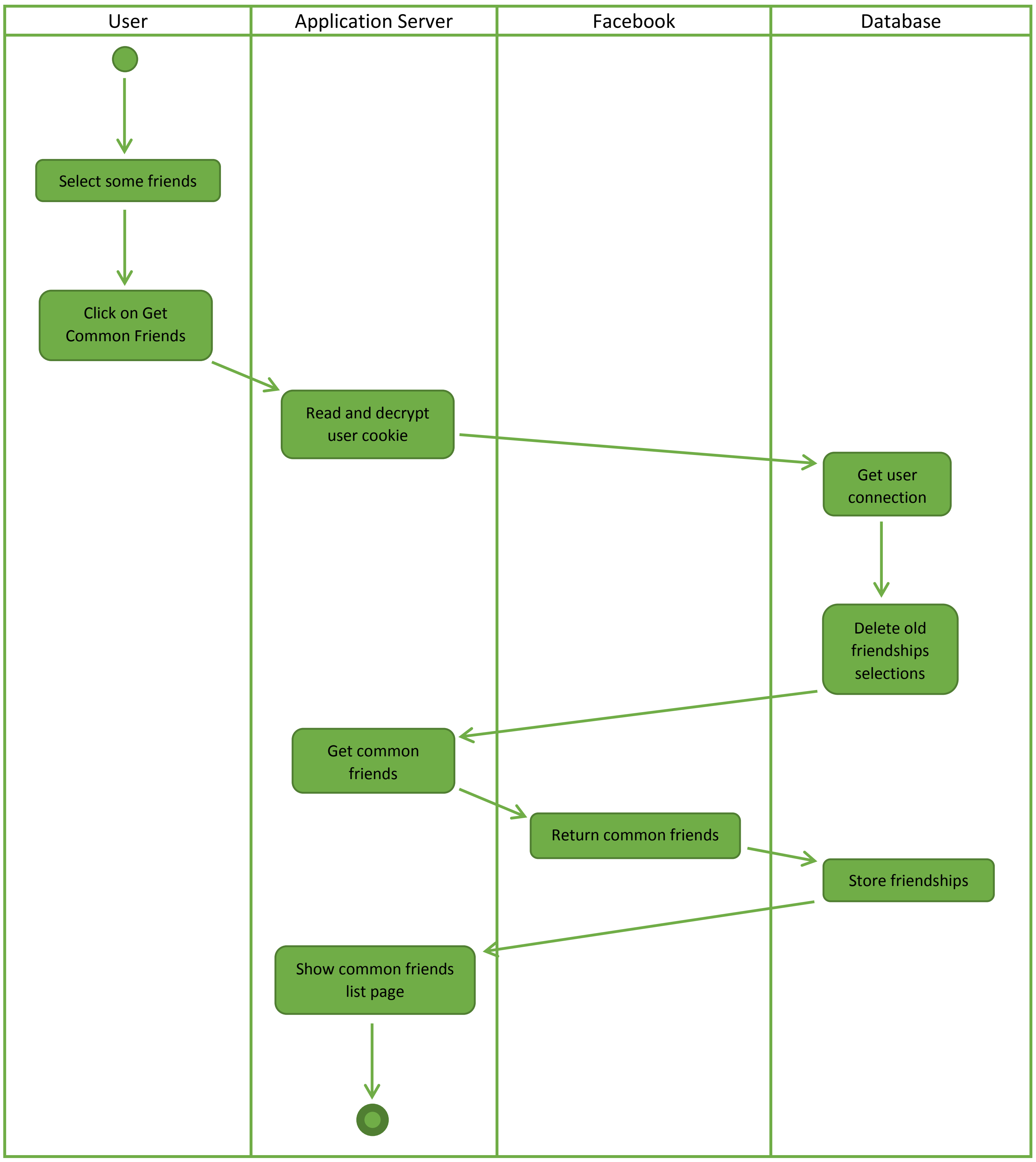


Get Friends List in Detail - Sequence Diagram

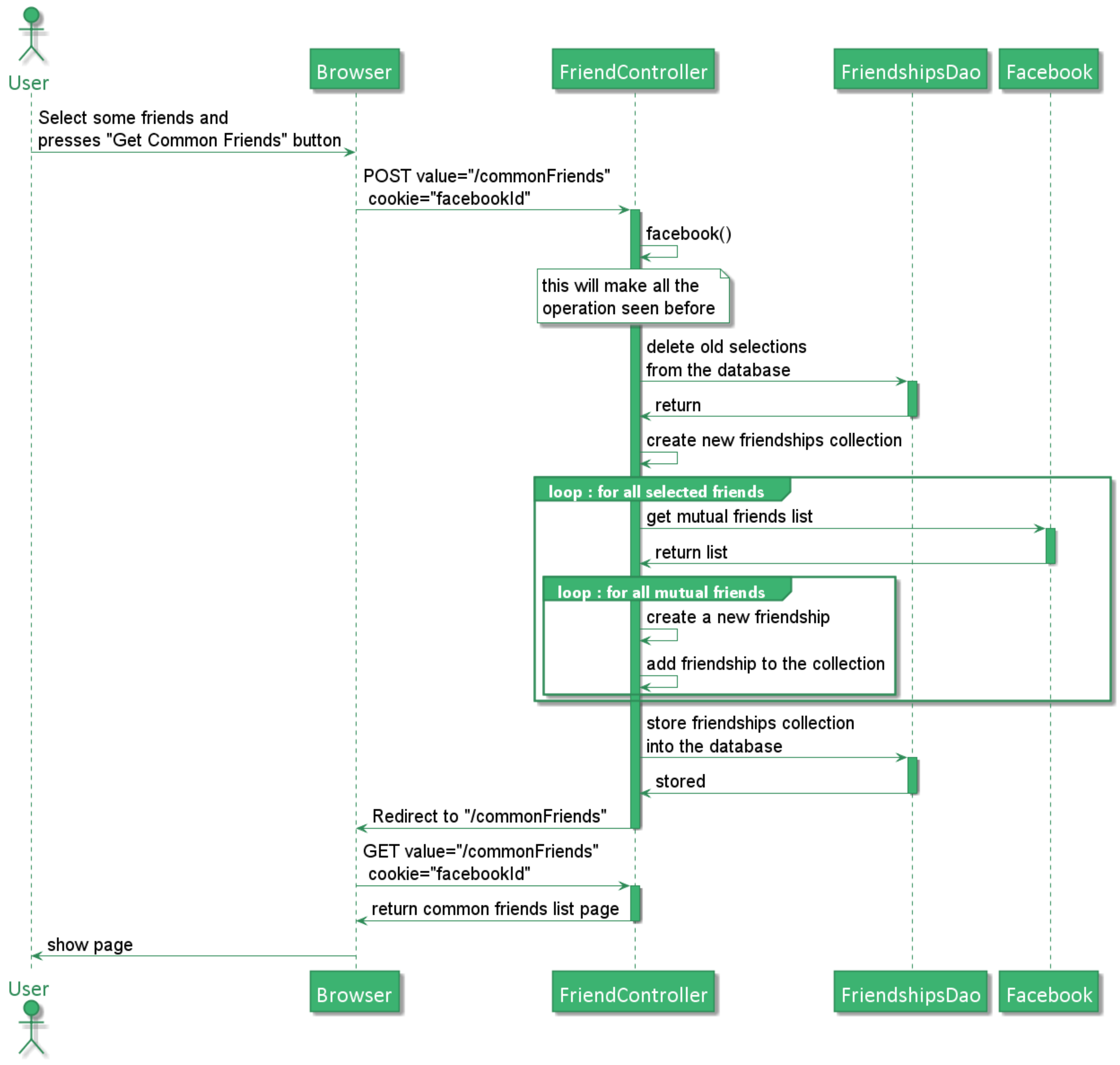


**Get Common Friends List - Activity Diagram**

After identify the user as before, the application first deletes from the database all the old stored friendship selections (created by previous interactions), in order to reduce garbage and also easily getting from the database only the useful friendships. Than it performs an API call in order to get all the common friends and then it stores them into the database. Finally it shows the common friends list page.

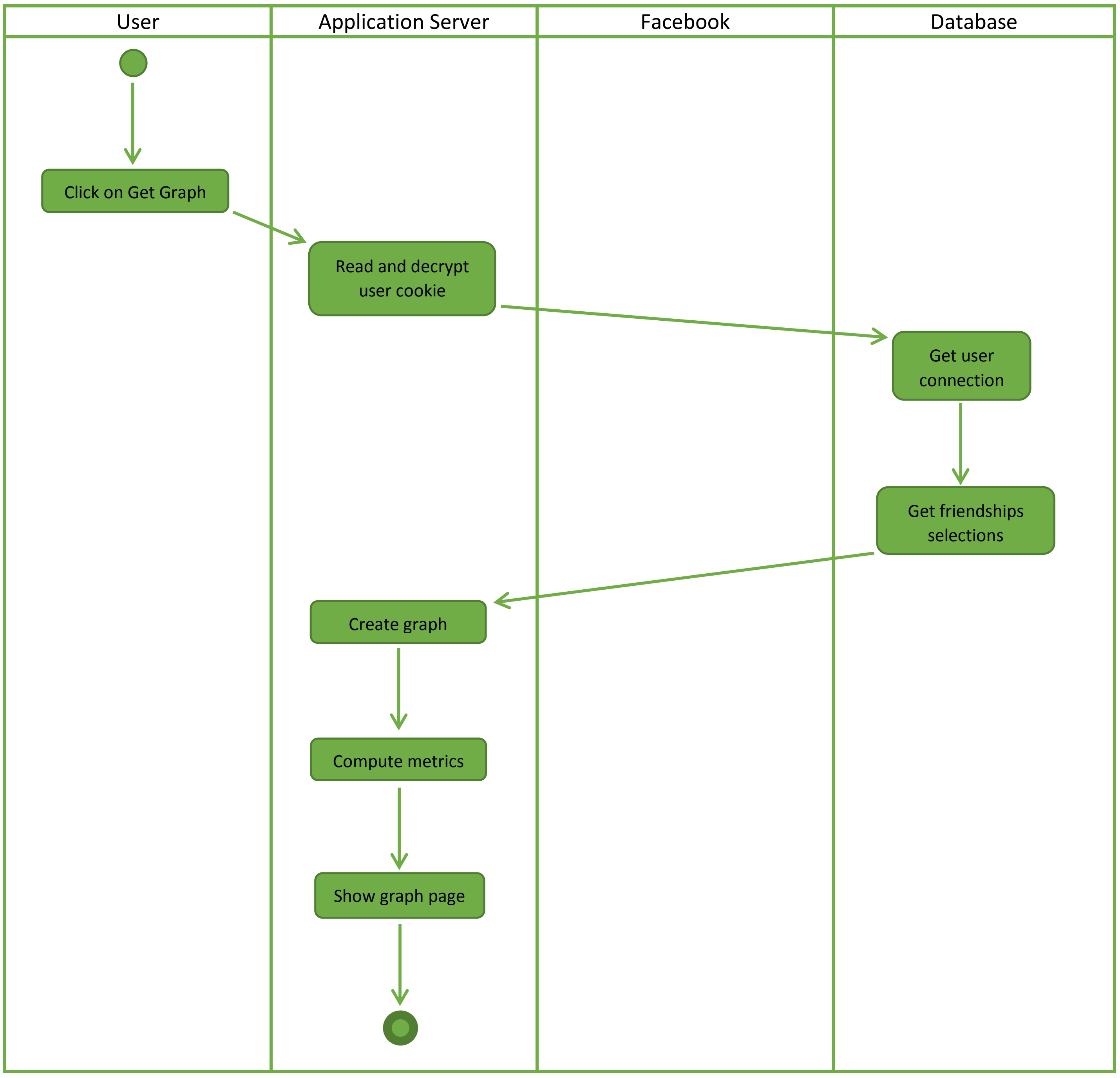


## Get Common Friends List in Detail - Sequence Diagram

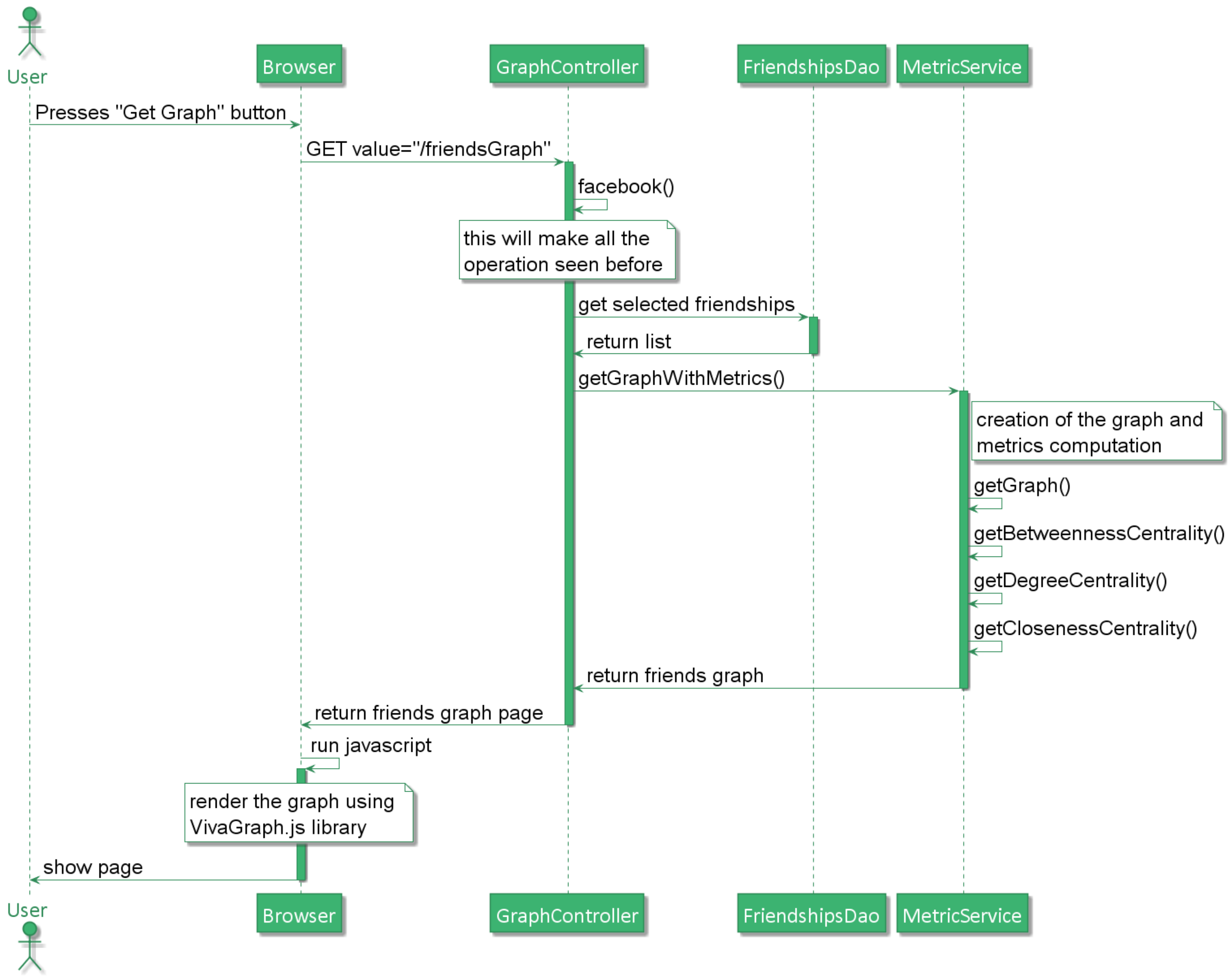

User
Browser
FriendController
FriendshipsDao
Facebook
Select some friends and presses "Get Common Friends" button
POST value="/commonFriends" cookie="facebookId"
facebook()
this will make all the operation seen before
delete old selections from the database
return
create new friendships collection
loop : for all selected friends
get mutual friends list
return list
loop : for all mutual friends
create a new friendship
add friendship to the collection
store friendships collection into the database
stored
Redirect to "/commonFriends"
GET value="/commonFriends" cookie="facebookId"
return common friends list page
show page
User
Browser
FriendController
FriendshipsDao
Facebook

**Get Friends Graph - Activity Diagram**

Once the user successfully retrieved the common friends from the one he selected, it can retrieve the friendship network graph. First it retrieves all the friendships from the database, then create the graph by setting all users as nodes and all friendship relations as edges. After that it computes all the metrics for each node and then it presents the page.



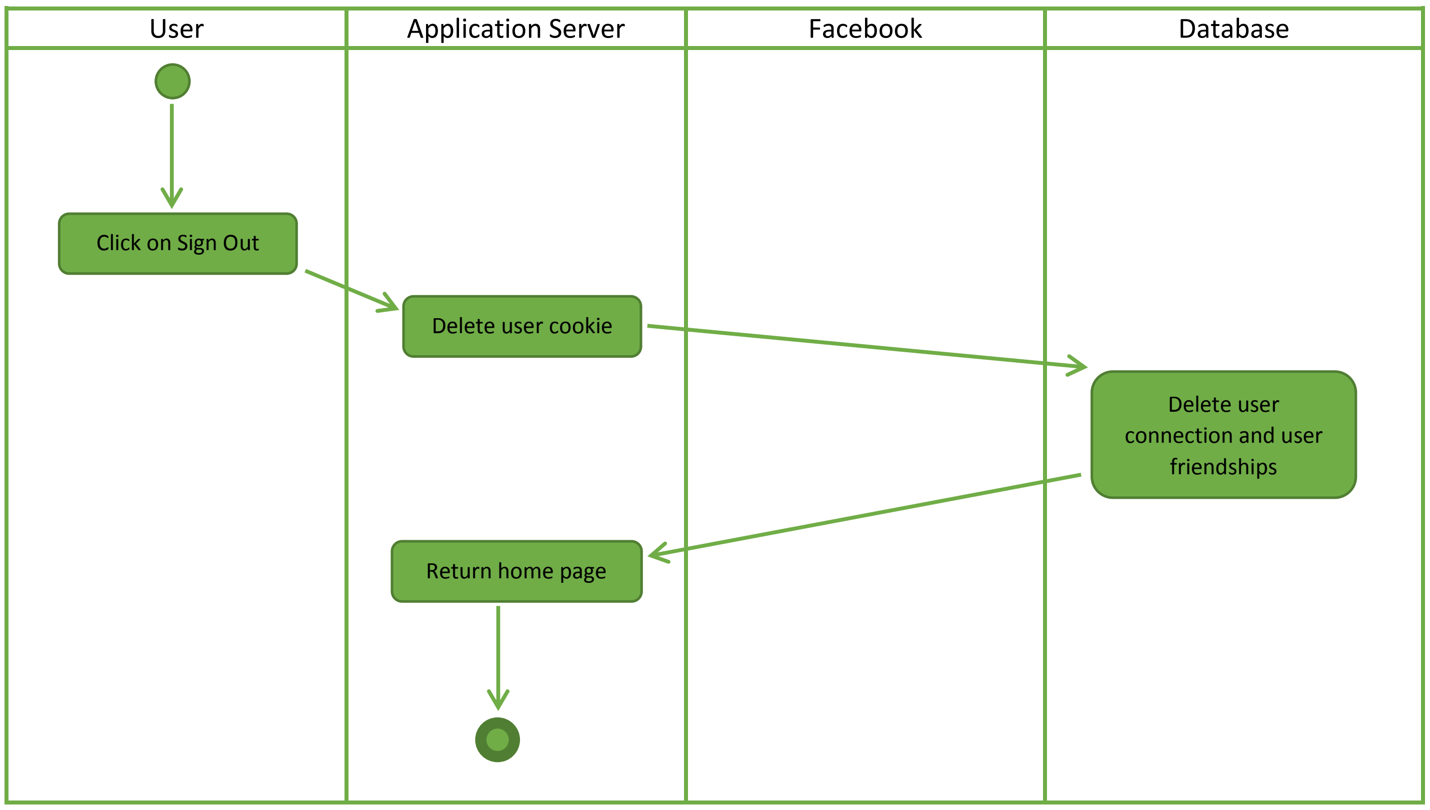
## Get Friends Graph in Detail - Sequence Diagram



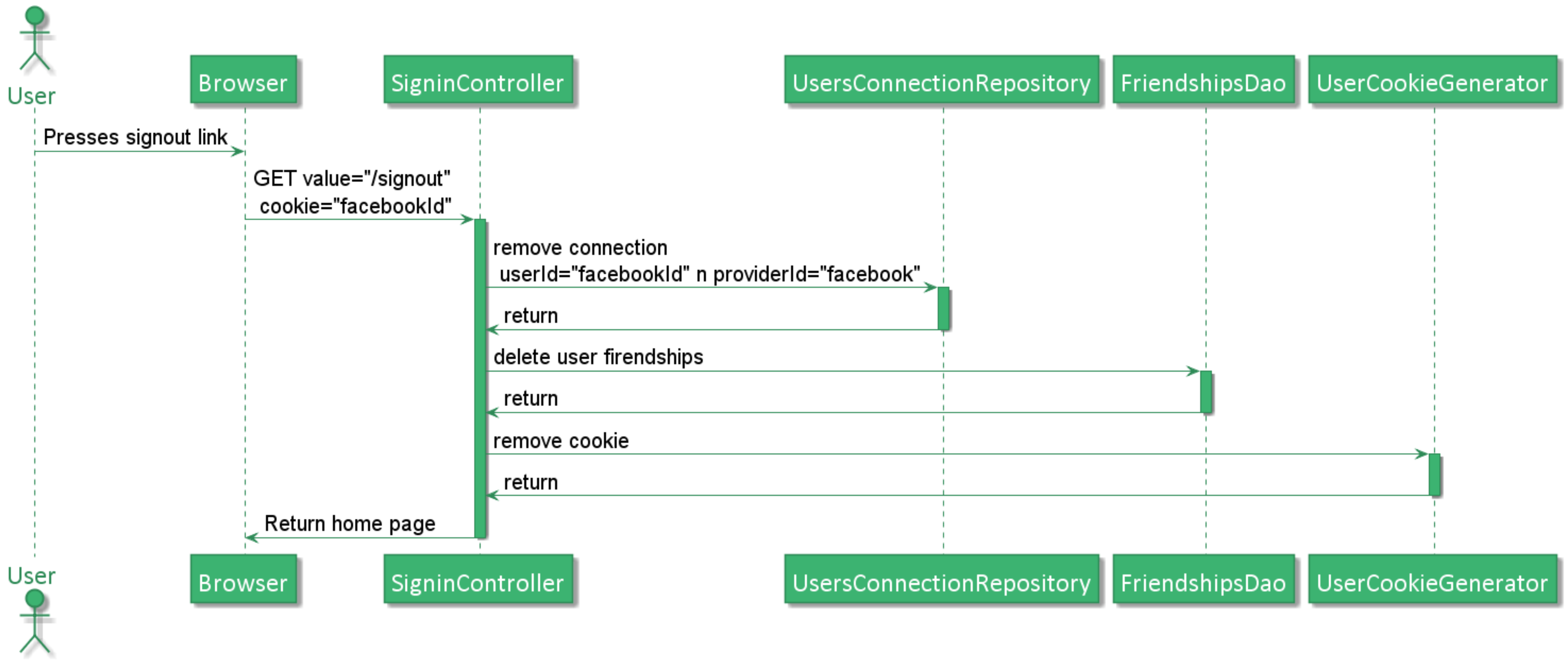
**User Sign-Out - Activity Diagram**

If the user want to sign out, the following operations are performed:

1. Deletion of user application cookie
2. Deletion of the user connection and all friendships created by his interaction
3. Back to the Home Page



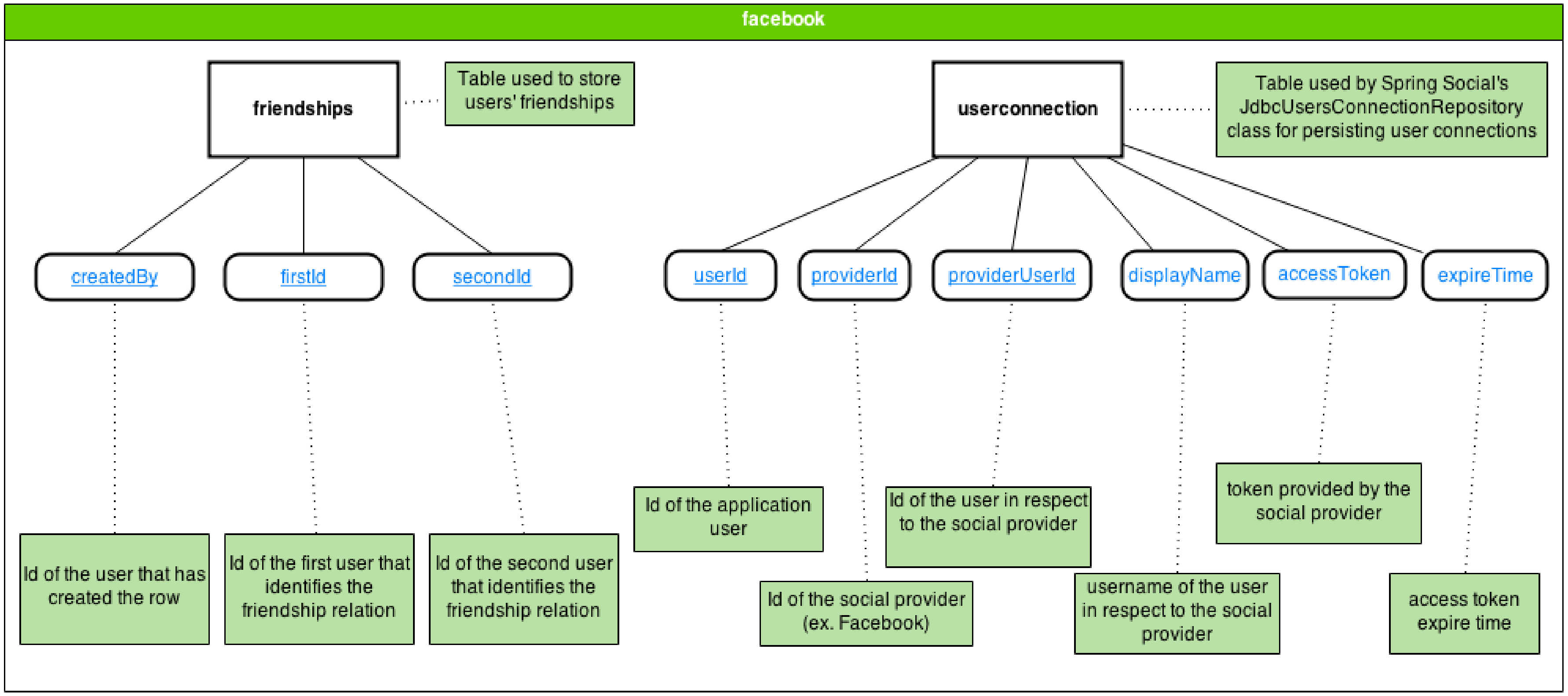
**User Sign-Out in Detail - Sequence Diagram**



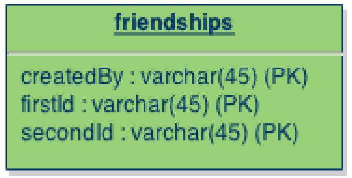
## Data Model Definition

We used the MySQL Database to store the information from the user's Facebook account. One table which is "userconnection" store the connections information related to the logged user. It keeps trace of the number of user connected to the web application and access token used to call the Facebook provider services. The other table called "friendships" contains the user's friendship relationship. The table is designed to provide the service to simultaneously connected users by saving information on who actually saved the data.

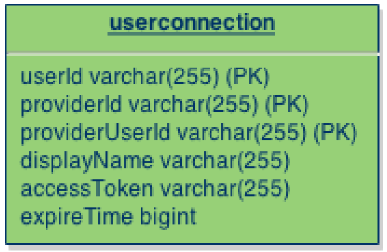
### Database Model


facebook
friendships
Table used to store users' friendships
userconnection
Table used by Spring Social's JdbcUsersConnectionRepository class for persisting user connections
createdBy
firstId
secondId
userId
providerId
providerUserId
displayName
accessToken
expireTime
Id of the user that has created the row
Id of the first user that identifies the friendship relation
Id of the second user that identifies the friendship relation
Id of the application user
Id of the social provider (ex. Facebook)
Id of the user in respect to the social provider
username of the user in respect to the social provider
token provided by the social provider
access token expire time

### Database Model – detail



| friendships |
|---|
| createdBy : varchar(45) (PK) |
| firstId : varchar(45) (PK) |
| secondId : varchar(45) (PK) |



| userconnection |
|---|
| userId varchar(255) (PK) |
| providerId varchar(255) (PK) |
| providerUserId varchar(255) (PK) |
| displayName varchar(255) |
| accessToken varchar(255) |
| expireTime bigint |

# Web Application Development

## Front-End Development

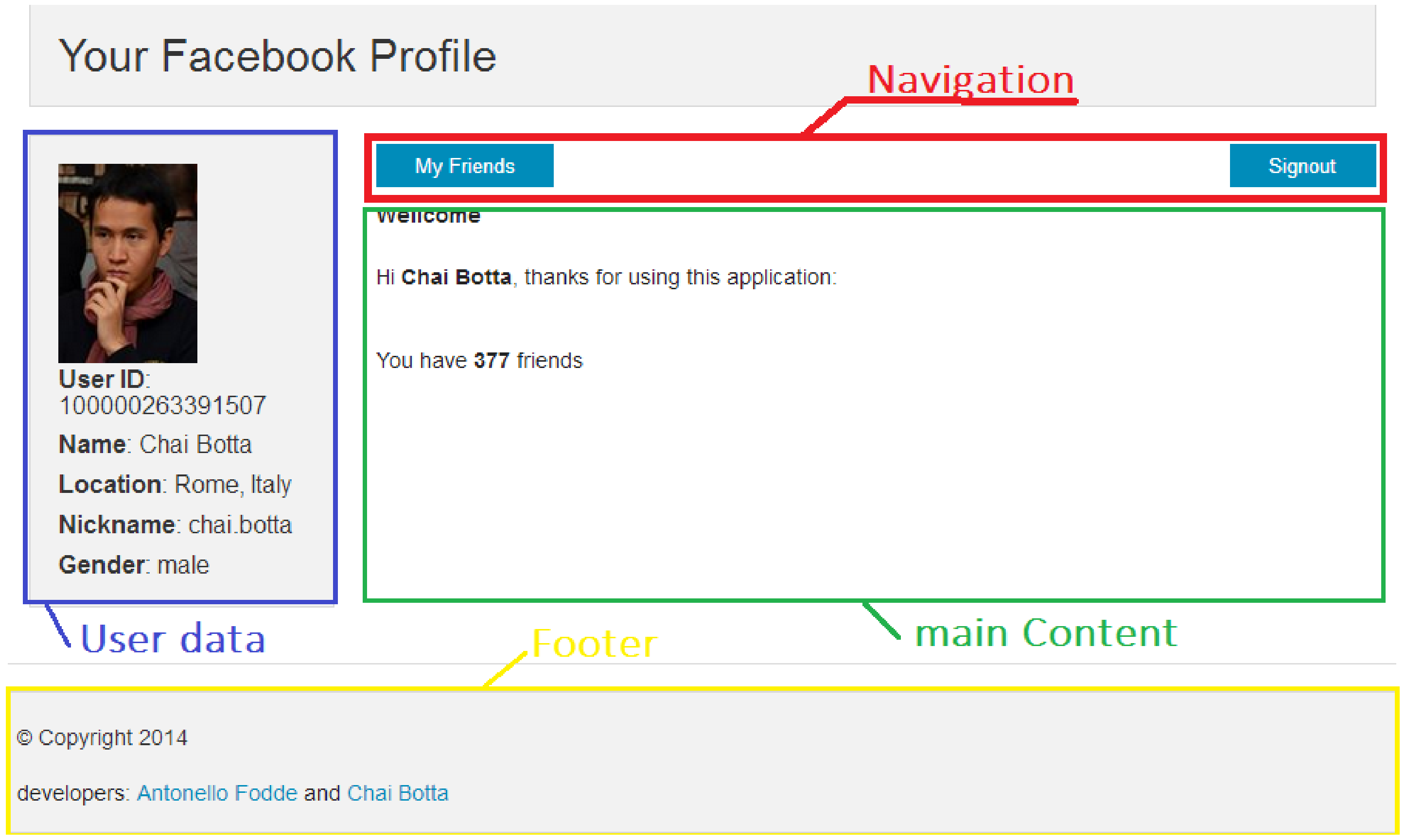
All the web-pages are dynamically content-created using JSP and JSTL taglib.

The User-Interface is done using the Foundation5 web-framework which allows to build easily nice and responsive web interfaces. The following .jsp pages (Views) are used to show the content (Model) required by the user.

- home.jsp
- friendslist.jsp
- commonFriends.jsp
- friendsGraph.jsp

The user can navigate sequentially from a page to another with exception to the commonFriends.jsp which required a friend selection from the friend list into the friendlist.jsp page.

Apart of the friendsGraph and the home the other pages follow a common template which is shown below:



User ID: 100000263391507
Name: Chai Botta
Location: Rome, Italy
Nickname: chai.botta
Gender: male

Wellcome

Hi **Chai Botta**, thanks for using this application:

You have **377** friends

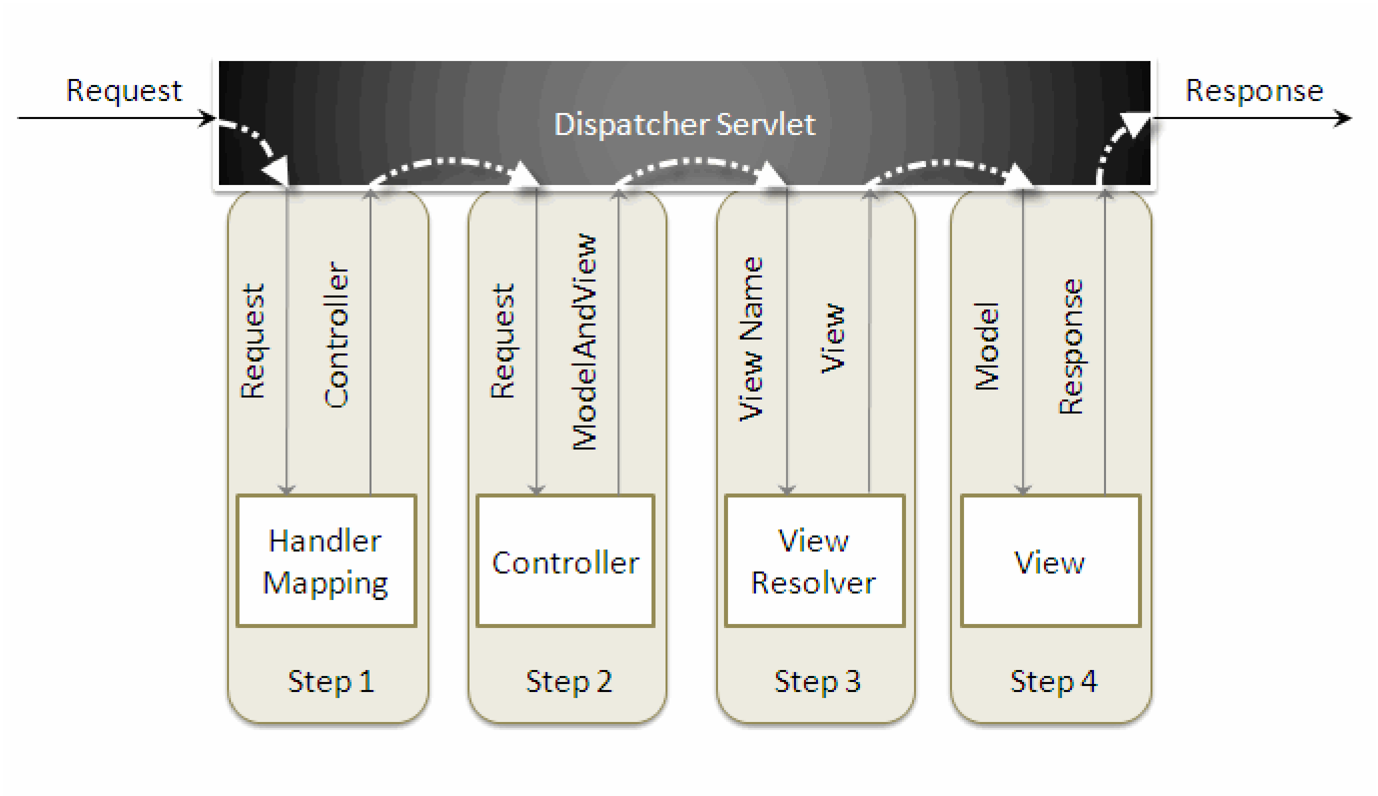
© Copyright 2014

developers: Antonello Fodde and Chai Botta

## Back-End Development

The back-end is managed by the Spring framework following the MVC pattern. The pages requested are mapped by a Spring class which is the DispatcherServlet which manages the Controllers. These last ones respond to the DispatcherServlet by selecting the page (View) after it's filled with the proper content (Model).

Here is how a request is managed by the DispatcherServlet into the Spring Framework.



The Controller defined for our application scope are:

- HomeController
- SigninController
- FriendsController
- GraphController

The HomeController is the first called by the DispatcherServlet and return the Homepage View.

The SigninController plays a central role into the application logic. It manage login/logout sessions and create the connection with the Facebook provider following the User Connection Sequence Diagram show previously.

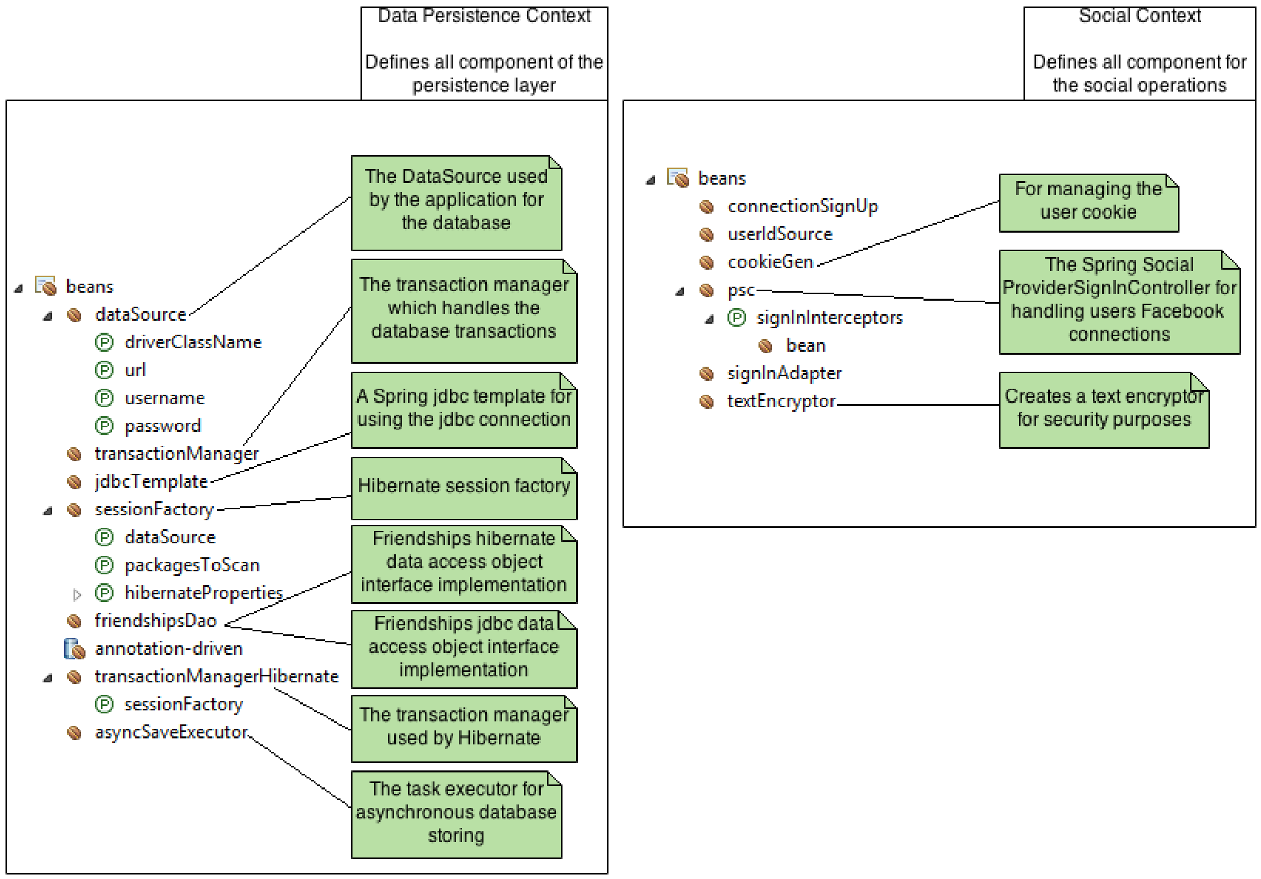
The FriendsController instead contains all the operations over the user's friends. It retrieves and returns the views with the list of friends and the list of common friends between the friends selected.

The last controller is the GraphController. Its only purpose is to compute operation over the friendships graph using the MetricService and it returns a view showing the graph. To render the visual graph we used an external Javascript library which is VivaGraph.

## Context Division

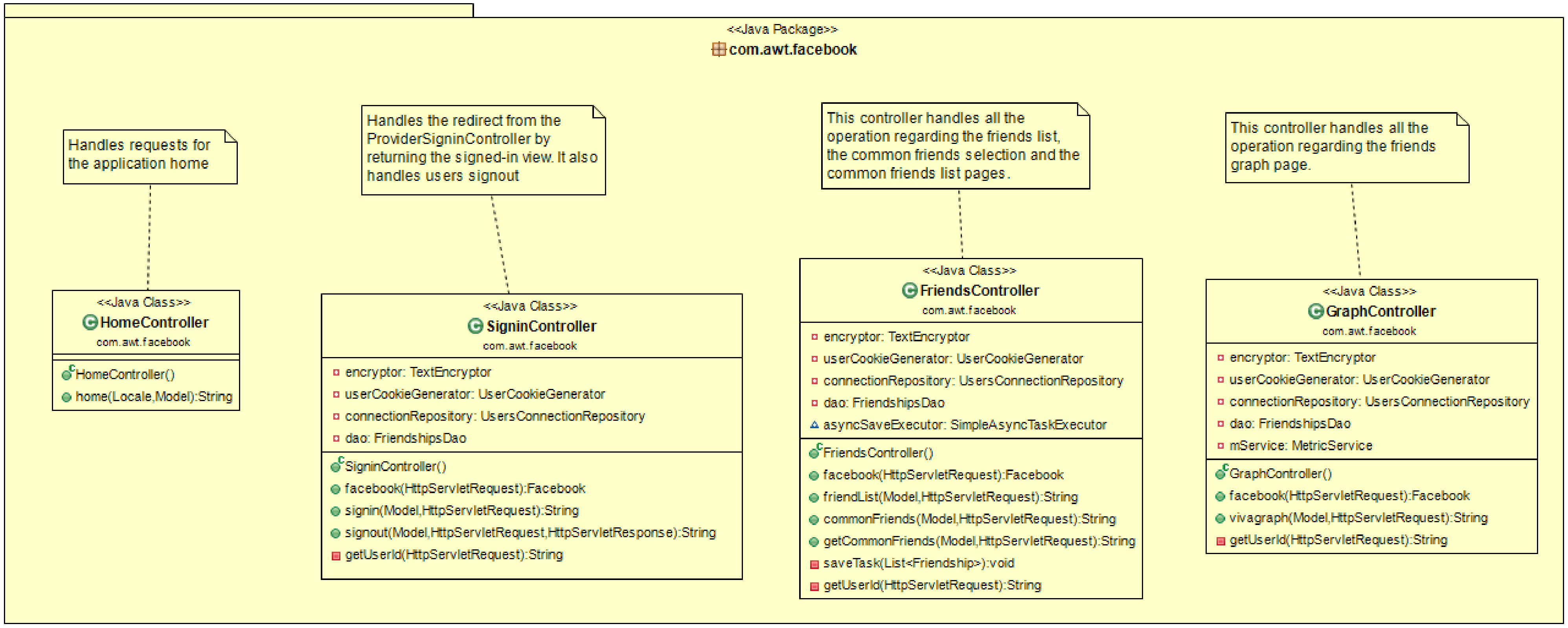
We preferred to adopt an XML-based approach instead to the traditional annotation-based one. Some of the advantages for this choice:

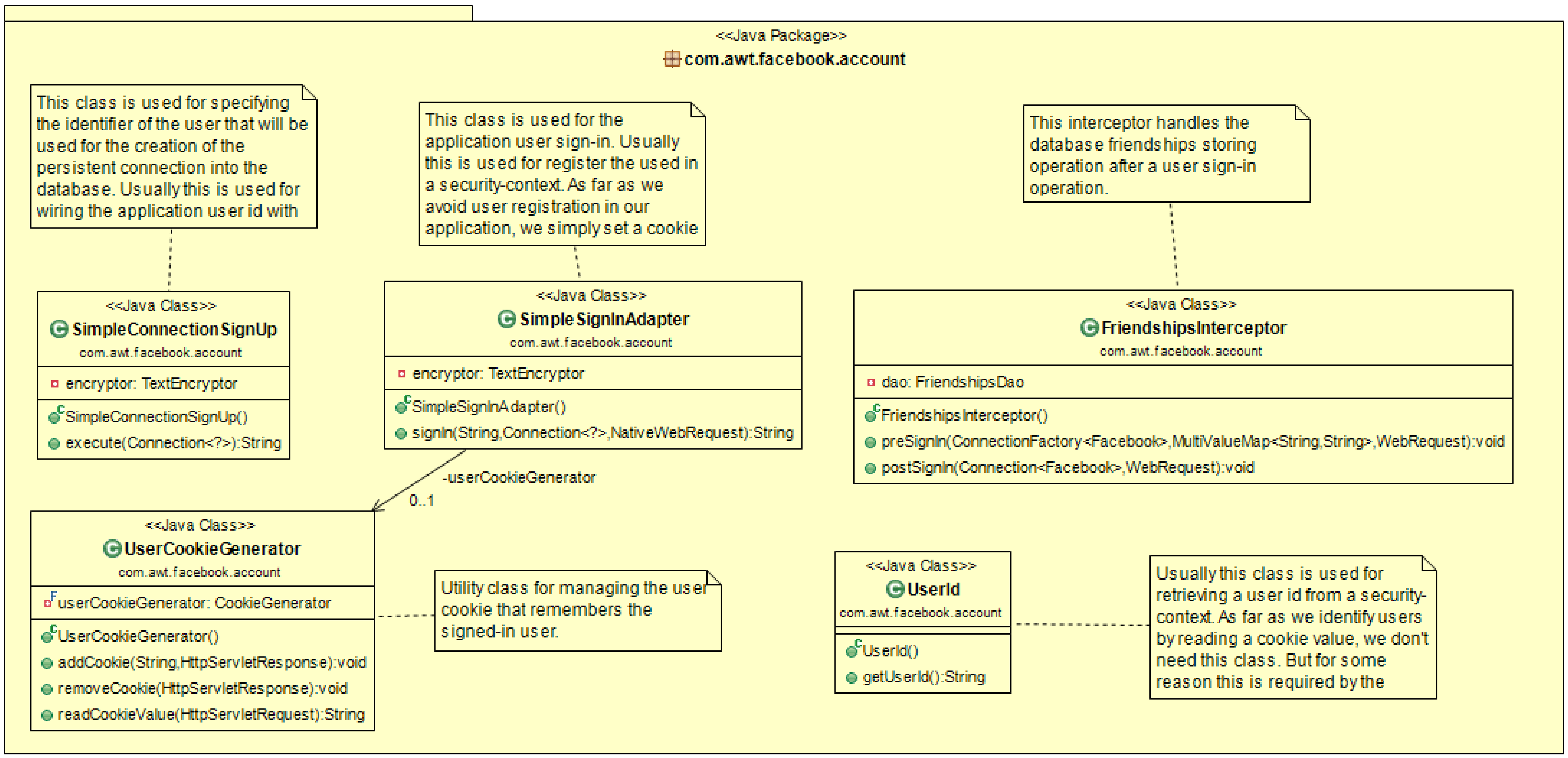
1. Configuration is centralized, it's not scattered among all different components so you can have a nice overview of beans and their wirings in a single place
2. If you need to split your files, no problem, Spring let you do that. It then reassembles them at runtime through internal <import> tags or external context files aggregation
3. Only XML configuration allows for explicit wiring – as opposed to autowiring. Its apparent simplicity hides real complexity: not only do we need to switch between by-type and by-name autowiring, but more importantly, the strategy for choosing the relevant bean among all eligible ones escapes but the more seasoned Spring developers. Profiles seem to make this easier, but is relatively new and is known to few
4. XML is completely orthogonal to the Java file: there's no coupling between the 2 so that the class can be used in more than one context with different configurations
5. Annotations quickly get out of control. Sometimes you see classes and methods with 4+ lines of annotations. It is not so nice.

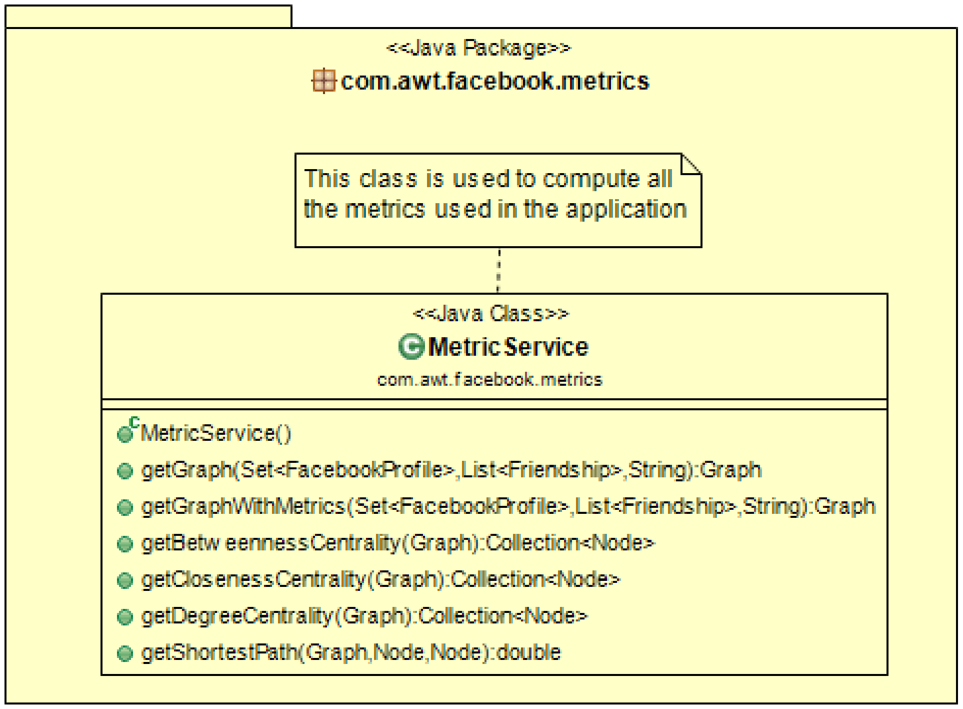

Data Persistence Context
Defines all component of the persistence layer
beans
dataSource
driverClassName
url
username
password
transactionManager
jdbcTemplate
sessionFactory
dataSource
packagesToScan
hibernateProperties
friendshipsDao
annotation-driven
transactionManagerHibernate
sessionFactory
asyncSaveExecutor
The DataSource used by the application for the database
The transaction manager which handles the database transactions
A Spring jdbc template for using the jdbc connection
Hibernate session factory
Friendships hibernate data access object interface implementation
Friendships jdbc data access object interface implementation
The transaction manager used by Hibernate
The task executor for asynchronous database storing
Social Context
Defines all component for the social operations
beans
connectionSignUp
userIdSource
cookieGen
psc
signInInterceptors
bean
signInAdapter
textEncryptor
For managing the user cookie
The Spring Social ProviderSignInController for handling users Facebook connections
Creates a text encryptor for security purposes

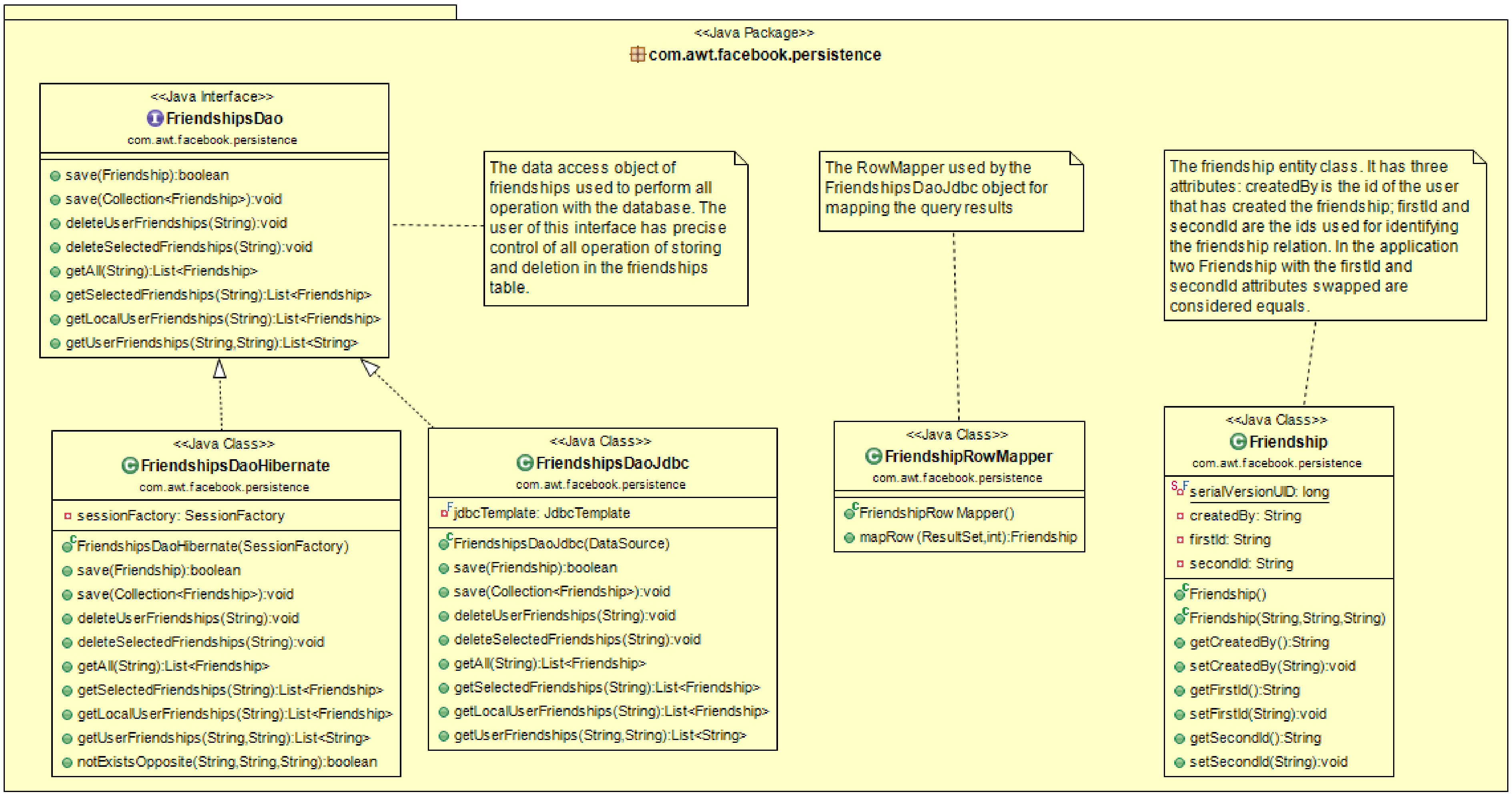
Note: more details later in the class diagrams

## Class Diagrams


<<Java Package>>
com.awt.facebook
Handles requests for the application home
Handles the redirect from the ProviderSigninController by returning the signed-in view. It also handles users signout
This controller handles all the operation regarding the friends list, the common friends selection and the common friends list pages.
This controller handles all the operation regarding the friends graph page.
<<Java Class>>
HomeController
com.awt.facebook
HomeController()
home(Locale,Model):String
<<Java Class>>
SigninController
com.awt.facebook
encryptor: TextEncryptor
userCookieGenerator: UserCookieGenerator
connectionRepository: UsersConnectionRepository
dao: FriendshipsDao
SigninController()
facebook(HttpServletRequest):Facebook
signin(Model,HttpServletRequest):String
signout(Model,HttpServletRequest,HttpServletResponse):String
getUserId(HttpServletRequest):String
<<Java Class>>
FriendsController
com.awt.facebook
encryptor: TextEncryptor
userCookieGenerator: UserCookieGenerator
connectionRepository: UsersConnectionRepository
dao: FriendshipsDao
asyncSaveExecutor: SimpleAsyncTaskExecutor
FriendsController()
facebook(HttpServletRequest):Facebook
friendList(Model,HttpServletRequest):String
commonFriends(Model,HttpServletRequest):String
getCommonFriends(Model,HttpServletRequest):String
saveTask(List<Friendship>):void
getUserId(HttpServletRequest):String
<<Java Class>>
GraphController
com.awt.facebook
encryptor: TextEncryptor
userCookieGenerator: UserCookieGenerator
connectionRepository: UsersConnectionRepository
dao: FriendshipsDao
mService: MetricService
GraphController()
facebook(HttpServletRequest):Facebook
vivagraph(Model,HttpServletRequest):String
getUserId(HttpServletRequest):String


<<Java Package>>
com.awt.facebook.account
This class is used for specifying the identifier of the user that will be used for the creation of the persistent connection into the database. Usually this is used for wiring the application user id with
This class is used for the application user sign-in. Usually this is used for register the used in a security-context. As far as we avoid user registration in our application, we simply set a cookie
This interceptor handles the database friendships storing operation after a user sign-in operation.
<<Java Class>>
SimpleConnectionSignUp
com.awt.facebook.account
encryptor: TextEncryptor
SimpleConnectionSignUp()
execute(Connection<?>):String
<<Java Class>>
SimpleSignInAdapter
com.awt.facebook.account
encryptor: TextEncryptor
SimpleSignInAdapter()
signIn(String,Connection<?>,NativeWebRequest):String
<<Java Class>>
FriendshipsInterceptor
com.awt.facebook.account
dao: FriendshipsDao
FriendshipsInterceptor()
preSignIn(ConnectionFactory<Facebook>,MultiValueMap<String,String>,WebRequest):void
postSignIn(Connection<Facebook>,WebRequest):void
-userCookieGenerator
0..1
<<Java Class>>
UserCookieGenerator
com.awt.facebook.account
userCookieGenerator: CookieGenerator
UserCookieGenerator()
addCookie(String,HttpServletResponse):void
removeCookie(HttpServletResponse):void
readCookieValue(HttpServletRequest):String
Utility class for managing the user cookie that remembers the signed-in user.
<<Java Class>>
UserId
com.awt.facebook.account
UserId()
getUserId():String
Usually this class is used for retrieving a user id from a security-context. As far as we identify users by reading a cookie value, we don't need this class. But for some reason this is required by the


<<Java Package>>
com.awt.facebook.metrics
This class is used to compute all the metrics used in the application
<<Java Class>>
MetricService
com.awt.facebook.metrics
MetricService()
getGraph(Set<FacebookProfile>,List<Friendship>,String):Graph
getGraphWithMetrics(Set<FacebookProfile>,List<Friendship>,String):Graph
getBetweennessCentrality(Graph):Collection<Node>
getClosenessCentrality(Graph):Collection<Node>
getDegreeCentrality(Graph):Collection<Node>
getShortestPath(Graph,Node,Node):double


<<Java Package>>
com.awt.facebook.persistence
<<Java Interface>>
FriendshipsDao
com.awt.facebook.persistence
save(Friendship):boolean
save(Collection<Friendship>):void
deleteUserFriendships(String):void
deleteSelectedFriendships(String):void
getAll(String):List<Friendship>
getSelectedFriendships(String):List<Friendship>
getLocalUserFriendships(String):List<Friendship>
getUserFriendships(String,String):List<String>
<<Java Class>>
FriendshipsDaoHibernate
com.awt.facebook.persistence
sessionFactory: SessionFactory
FriendshipsDaoHibernate(SessionFactory)
save(Friendship):boolean
save(Collection<Friendship>):void
deleteUserFriendships(String):void
deleteSelectedFriendships(String):void
getAll(String):List<Friendship>
getSelectedFriendships(String):List<Friendship>
getLocalUserFriendships(String):List<Friendship>
getUserFriendships(String,String):List<String>
notExistsOpposite(String,String,String):boolean
<<Java Class>>
FriendshipsDaoJdbc
com.awt.facebook.persistence
jdbcTemplate: JdbcTemplate
FriendshipsDaoJdbc(DataSource)
save(Friendship):boolean
save(Collection<Friendship>):void
deleteUserFriendships(String):void
deleteSelectedFriendships(String):void
getAll(String):List<Friendship>
getSelectedFriendships(String):List<Friendship>
getLocalUserFriendships(String):List<Friendship>
getUserFriendships(String,String):List<String>
<<Java Class>>
FriendshipRowMapper
com.awt.facebook.persistence
FriendshipRowMapper()
mapRow(ResultSet,int):Friendship
<<Java Class>>
Friendship
com.awt.facebook.persistence
serialVersionUID: long
createdBy: String
firstId: String
secondId: String
Friendship()
Friendship(String,String,String)
getCreatedBy():String
setCreatedBy(String):void
getFirstId():String
setFirstId(String):void
getSecondId():String
setSecondId(String):void

The data access object of friendships used to perform all operation with the database. The user of this interface has precise control of all operation of storing and deletion in the friendships table.

The RowMapper used by the FriendshipsDaoJdbc object for mapping the query results

The friendship entity class. It has three attributes: createdBy is the id of the user that has created the friendship; firstId and secondId are the ids used for identifying the friendship relation. In the application two Friendship with the firstId and secondId attributes swapped are considered equals.

# Conclusion

## Lessons learned

Developing this web application using a Framework such as Spring was a great challenge. It took a while to learn the new paradigm introduced by the Framework such as the huge use of annotation/configuration file via xml and the injection methodologies. It took us more than two months of self-study before we started to manage proficiently with the Framework philosophy included the use of Maven as project builder.

What we learned doing the project are:

1. Using Frameworks can be really painful at the beginning due to the learning-curve

2. But using Frameworks can really speedup development phase for any software project (starting from the second)

3. New paradigms are aimed to improve codes and logic

4. Frameworks help you to focus more on business logic than on how/what is at the background

5. Interface is what the user get, so try to do it user-friendly and good looking (it is what matter most to the user)

## Future works

In a next future we would like to use part of the application to build a serious-backend for some html5 canvas games and build up a REST API service to let mobile application to connect to the web application.
It could also be interesting to improve the interaction with the application: currently it is uncomfortable after generating a graph, going back to the selection page to make another one. One solution could be adding the possibility of dynamically creating the graph directly from the page of the same, in response to a selection of a node. For a more clear idea see this example.

## References

| Name | Description | Link |
|---|---|---|
| Foundation5 | **ZURB Foundation** is a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional Javascript extensions | http://foundation.zurb.com/ |
| Spring Framework | Open source application framework and inversion of control container for the Java platform. | https://spring.io/ |
| Hibernate ORM | Object-relational mapping library for Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. | http://hibernate.org/ |