Design Document

Florida International University

**Virtual job fair v5.0**

CIS 4911 – Senior Project

**mEMBERS**:

Rogelio alonso

**INSTRUCTOR**:

MASOUD SADJADI

**MENTOR:**

JUAN CARABALLO

2015

Copyright and trademark notices, restrictions on copying or distributing the documentation, information for contacting the issuing organization (reader’s comments), warranties, contractual obligations or disclaimers, and general warnings and cautions.

**ABSTRACT**

The Design Document gives an insight into the system structure of the Virtual Job Fair 5.0 project. By describing system architecture, subsystem decomposition, design methodology, hardware and software mapping, persistent data management and security/privacy gives the reader a better idea of the process that was used to design Virtual Job Fair 5.0. The first part of this document talks about the problem definition and gives an overview of the document and what it’s going to cover. On part two, this document introduces the structure of the system by describing its system design, subsystem, and provides models and diagrams for better understanding.

Part 3 goes into detail design, which discusses static models and dynamic models plus the specification for the class interface. Part 4 of the Design Document contains a glossary which explains the domain-specific terms, as well as miscellaneous information, such as use case diagrams, use cases implemented, and the diary of meetings and tasks.

Table of Contents

1.Introduction 4

1.1. Problem definition 4

1.2.Design methodology 5

1.3. Definitions, acronyms, and abbreviations. 6

Definitions: 6

1.4. Overview of document 7

2. System Design 8

2.1. Overview 9

2.2. Subsystem Decomposition 10

2.3. Hardware and Software Mapping 16

2.4. Persistent Data Management 17

2.5. Security/Privacy 21

3. Detailed Design 22

3.1. Overview 22

3.2. Static model. 26

3.3. Dynamic model – state machine diagram for the main control object in each subsystem. Include the design of the main algorithms used in the problem solution. 30

3.4. Code Specification - describe the class interfaces (attributes and method signatures) and constraint (invariants, pre-condition and post-conditions) for the main control object in each system. 35

4. Glossary 37

5. Appendix 39

5.1. Appendix A - Use case diagram for use cases being implemented. 39

5.2. Appendix B - Use cases being implemented. 40

5.3. Appendix C – Documented class interfaces (code) for the subsystem(s) you will implement and the constraints. 48

5.4. Appendix D - Diary of meeting and tasks. 50

6. References 53

# Introduction

      The introductory chapter gives some background information about the Virtual Job Fair v5.0 system. Firstly, the chapter states the problem definition, and scope of the system. Next, the design methodology used is identified. This methodology includes the software process models and the types of models used. Moreover, definitions, acronyms, and abbreviations of terms that will be used in this deliverable are introduced and explained. Finally, it contains an overview of the whole project, which explains the information contained on each chapter.

## . Problem definition

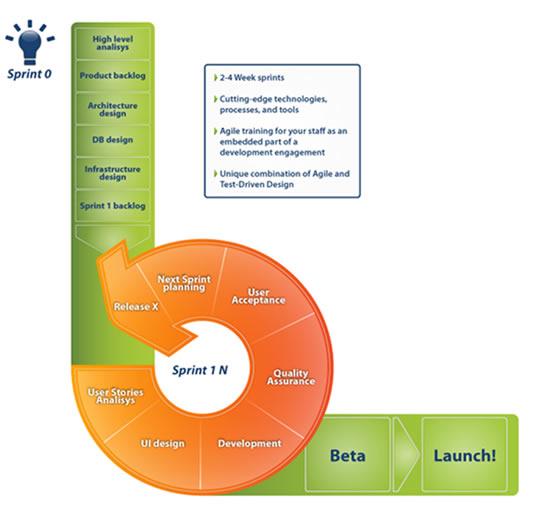
Employers looking for talent are always interested in filling out positions with the best possible people. In order to accomplish this task, the most effective method to date is to tap local talent, whether it is at universities or job fairs. Given the increasing globalization trend, and the fact that not all employers have the financial or logistical capabilities to seek for potential candidates in different locations, the current solution provided by universities and job sites is less than ideal.

Our solution to this problem, will be to provide FIU Computer Science students with a user-friendly interface where students will be able to get job matches from different job search sites, and not just from the employers who register in the VJF site. This way the students do not need to go looking in different sites, and he or she can find everything on our site. Also students will receive notification of jobs matching his/her saved search criteria, so that way the students will know as soon as an opportunity is available. We will also provide a way for employers to search for specific skills on the students, as well as save queries of their own which they can be notified according to whichever time they want. Therefore providing the employer and potential employee a more easy and friendly way to find their match. Moreover, we will grant access to any visitor as an Employer or a Student Guest with limited functionality.

## .Design methodology

Agile Development Methodology was used in order to add the necessary functionality. Since Agile is iterative and incremental; we were developing functionalities and testing it along the way. It is very important to follow the rule imposed by this methodology in order to deliver a quality product that conforms and exceeds all standards.

We have found the iterative process to be very simple since it allows us to make small but concise progress towards the completion of the project. Below is the Agile Development Methodology diagram which was followed for the completion of this project.



For Virtual Job Fair V 5.0 we used Mingle in order to track and document the projects progress and requirements. Mingle allowed us to break the user stories into task, and then track the progress of all tasks in order to complete a user story. For the sake of clarity, in this version we created cards representing user stories from what was leftover on Trello from previous terms. We placed this cards under the Product Backlog card in Mingle. New user stories (Not refined or accepted by Product Owners), where place under the Product Backlog card. On every Sprint planning, accepted user stories (cards) by the product owner were assigned to the specific sprint. After this, these card tasks went through mockup To Do, In Dev, Testing and Done. Mingle board also has a feature so that you can see who the card belongs to and murmurs made in reference to specific cards.

## 1.3. Definitions, acronyms, and abbreviations.

## Definitions:

- **Student**: an individual who is currently enrolled in the School of Computing & Information Sciences FIU

- **Job:** an activity done in exchange for payment

- **Full-time:** requiring 40 hours or more hours per week

- **Part-time:** requiring less than 40 hours per week

- **Paid internship:** an internship for which a student will receive compensation

- **Unpaid internship:** an internship for which the student will not receive compensation

- **Benefits:** non-salaried compensation for employees, such as insurance, tuition reimbursement, and retirement benefits

- **Work authorization:** current legal work status of a student. Categories include U.S. Permanent Resident and U.S. Citizen

- **Grade point average:** a number out of 4.0 which gives a representation of a student’s grades in his/her classes throughout his/her college career

**Acronyms**

- **VJF**: Virtual Job Fair

- **FIU:** Florida International University

- **GPA:** Grade point average

- **SCIS:** School of Computing & Information Sciences

- **USDP:** Unified Software Development Document

**Abbreviations**

As of right now, there are no abbreviations for this project.

## 1.4. Overview of document

In chapter 1, the main problem is introduced, along with the design methodology used for the project, definitions, acronyms and abbreviations. In chapter 2, the system is introduced in terms of system architecture, with its subsystem decomposition, hardware and software mapping, persistent data management and security/privacy aspects explained. In chapter 3, the behavior of each subsystem is described, and the static models and dynamic models used are explained.

In chapter 4, a glossary with domain-specific terms is introduced. In the appendix, miscellaneous material, such as use case diagrams, use cases being implemented and documented class interfaces can be found. Finally, a diary of meetings and references can be found at the end of the document.

# 2. System Design

This chapter gives an insight into the architectural pattern(s) used to build the system. Virtual Job Fair was subdivided into subsystems, each one with a specific functionality that adds richness to the interview process. In this chapter, an overview of the system design is introduced. Then, the decomposition of the system into subsystems is explained. Moreover, hardware and software mapping and persistent data management aspects of the project are discussed. Finally, the security and privacy issues of the system are introduced.

The previous system shall…

Allow Guest Employers to access the system.

Allow Guest Students to access the system.

Allow Guest Employer to search for students based on one skill.

Allow Guest Employer to see student profiles.

Allow Guest Employer to post jobs.

Allow Guest Student to search for job postings.

Allow Students to replace their PDF resume.

Allow Students to replace their video resume.

Allow Students to upload a video resume of up to 80 MB.

Allow Students to replace their profile picture.

Allow future developers to test for regression changes through automated test suite.

Allow administrator to delete student accounts.

Allow administrator to delete employer accounts.

Provide a more reliable experience throughout the system.

Provide future developers with a search engine to improve the search algorithms.

The previous system shall…

Allow administrator to create student account

Allow administrator to create employer account

Allow administrator to view website and user statistics.

Allow student to search saved queries from advance search.

Allow student to search multiple saved queries at the same time from advance search.

Allow student to run search on advance search form and saved query at the same time.

Allow student to receive notifications on saved queries dynamically.

Allow employers to search for a job based on skill, location, zip code, school, major, graduation date, experience and position on the advance search.

Allow employers to save queries.

Allow employers to search saved queries form advanced search.

Allow employers to search multiple saved queries at the same time form advanced search.

Allow employer to run search on advance search form and saved query at the same time.

Allow employer to receive notification on saved queries dynamically.

Allow system to retrieve job postings from stackoverflow.com

Allow system to retrieve job postings from monster.com

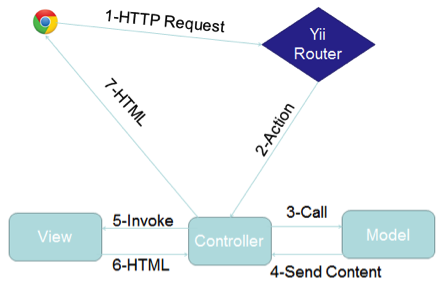
Allow system to retrieve jobs posting from jobs.github.com

Embed search algorithm of Employer Navigation Bar into the current search subsystem.

## 2.1. Overview

The previous team implemented Virtual Job Fair using the Model-View-Controller architecture. Our team is adding functionality to the system, meaning that the Model-View-Controller architecture will be maintained, with models, views and controllers added to account for the new functionality. Below is their description of the architecture of the system:

“The Architectural pattern used to build the system was the popular Model-View-Controller. This is the architecture implemented by the Yii Framework which was used by the team of developers. The following diagram depicts the architecture:”



The model represents the data, and does nothing else. The model does NOT depend on the controller or the view.

The view displays the model data, and sends user actions (e.g. button clicks) to the controller. The view can be independent of both the model and the controller; or actually be the controller, and therefore depend on the model.

The controller provides model data to the view, and interprets user actions such as button clicks. The controller depends on the view and the model. In some cases, the controller and the view are the same object.

## 2.2. Subsystem Decomposition

The previous group had a total of 8 subsystems, while we didn’t add any new subsystems we will be showing our modifications and additions to the subsystems.

**Screen Share subsystem**

Additionally the video screen Share subsystem allows users to share their screens with one another. This functionality is facilitated by the Screen Leap API. Screen Leap is a service that provides screen sharing functionality without the need for any of the users to install any specific software all that is required to function is the latest version of Java for the broadcaster. Screen Leap is able to function perfectly with the system by providing different API calls that allow for easy sharing of information. By eliminating the need for the user to download extra software we are able to facilitate and simplify the usage of the Virtual Job Fair website.

VJF-042 Share Screen

VJF-043 View Screen Share

VJF-044 End Screen Sharing

**Whiteboard subsystem**

The whiteboard subsystem will allow users to draw on a whiteboard during a live interview. The whiteboard includes drawing, writing, erasing, clearing and picture-saving functionality, which makes it an ideal tool to share images during a live interview, to jot down notes or simply to brainstorm. The use cases related to the whiteboard are:

VJF-060 Show or Restore Whiteboard

VJF-062 Draw With Pencil

VJF-063 Change Drawing Tool Pencil

VJF-064 Type Text Into Whiteboard

VJF-065 Clear contents of whiteboard

VJF-066 Erase From Whiteboard

**Video Interview Subsystem**

The Video Interview Subsystem allows users to have a virtual interview. This is possible thanks to Web-RTC technology. Web-RTC is an open source project that allows web browsers to communicate directly with each other with the aid of Java Scrip API calls and HTML5. This new technology makes the communication between internet users easier than traditional methods. Thanks to Web-RTC one can share video feed with other users without the need for media servers or plug-ins.

The synchronization of video interviews is handled in a table in the database. This is very important because we need to make sure only users scheduled to have an interview can be in the interview page. This is done by using a session key which is unique for each video interview scheduled. Users that arrive at the interview page would do so via a link which has many parameters such the session key. Once both users arrive at the interview page, the application will check for the session key parameter and match them, and only those users that have matching session keys would be able to connect to each other.

The uses cases related to this subsystem are:

· VJF-0020 Start Video Interview

· VJF-0021 Accept Interview

· VJF-0030 Schedule Video Interview

**E-mail Message Subsystem**

The E-mail messaging subsystem allows users to keep in contact and communicate with each other right on the system. It is very similar to a traditional inbox, only that it is internal to the system, similar to LinkedIn’s messaging. The messaging subsystem uses database tables to store and retrieve messages sent between users. It allow employers to message students and in doing so open up a line of communication with them; i.e., a student is able to message an employer only after the employer has initiated the communication with that particular student. Storing and retrieving messages efficiently is imperative to a successful messaging system. The messaging subsystem uses AJAX to rapidly access and store data; allowing users to interact faster with the system without having to wait for server calls.

The use cases related to this subsystem are:

Ø VJF-0022 Reply to Message

Ø VJF-0023 Send Message

**Notification Subsystem**

The subsystem is what allows users to stay up to date with the latest system interactions. The notification subsystem alerts users of any interaction by other users that might implicate them, such as a video interview been schedule for a user, or a new job post that matches a user’s skills.

The notification subsystem relies on the database structure to be able to efficiently keep the users inform. Due to the relationships between tables in the database the notification subsystem can easily detect what notification belongs to what user. Also it is important to sort each notification by category and level of importance. The notification subsystem achieves this by storing different types of notifications in the database and mapping them to their respective categories.

**Automated Notification**

This subsystem make use of the “job matching notification engine”. It is in charge of sending notification emails with the job listings to the students based on jobs matching their skills or based on customized saved query preferences. Students and employers can now get notifications in between any period of time they want. They job listing receive in the email will contain jobs from outside sources like Indeed.com, and CareerBuilder.com, as well as the job posting from the database which will contain jobs from FIU CareerBuilder. And for the employer it will be a list of the student that match the saved query description.

The uses cases related to this subsystem are:

Ø VJF-0021 Accept Interview

Ø VJF-001 Registration

Ø VJF-0034 Read notification

Ø VJF-059 Set Notifications On

Ø VJF-064 Admin Enable Notification

**Student Profile subsystem**

Being able to create a good profile fast and efficiently is very important. The profile creation subsystem takes care of this by allowing students to import profile information from third party websites such as LinkedIn. This ensures integrity of the data in students’ profiles, and makes it very easy for students to create their profiles.

This is possible by using API calls to LinkedIn and retrieving the data from LinkedIn users. As it is to expect, the user must grant permission to do this by providing his/her login credentials which are handled by the LinkedIn API.

**Merge Account**

This subsystem provides students with a view form to input the username and password of the other account. Then validate the information given, and if the information is verified, merge the two accounts by comparing the student information that he or she has in the database. And give the student user the choice to keep the information that he/she wants when a merge conflict happens.

**Linking Account**

This subsystem provides students with a way to link all third party account into one account. Also, it lets the student know if they are linked to a third party account or not, and which one. When linking, students can choose which information to keep if there is a conflict among information.

The use cases related to this subsystem are:

Ø VJF-001 Registration

Ø VJF-0019 Integrate LinkedIn

Ø VJF-003 Edit Basic Info

Ø VJF-004 Verify Email

Ø VJF-008 Edit Picture

Ø VJF-009 Upload Resume

Ø VJF-0011 Add Education

Ø VJF-0012 Delete Education

Ø VJF-0013 Add Experience

Ø VJF-0014 Delete Experience

Ø VJF-063 Linking Account Google

Ø VJF-067 Merge Account

**Student job match subsystem**

Making the right connection is what this web application is all about. Therefore, an efficient algorithm to match students to job openings is very important. The student job match subsystem takes care of matching students with the required skills to job post, making the job of the recruiters easier, as it shrinks the search to only the most qualified individuals for the job.

The student job match subsystem relies on the relationships between the data in the job table. By matching job skills to students skills listed on their profile the algorithm can effectively narrow down the search to only those individuals who possess those skills.

The uses cases related to this subsystem are:

Ø VJF-0016 Add Skill

Ø VJF-0017 Delete skill

Ø VJF-0018 Change skills Order

**Search Subsystem**

The search subsystem is compose of the key features of the job search, which allows students and employers to perform advanced search, navigation bar search, as well as save queries to later receive email notifications with job results. This subsystem make the use of job search possible, giving the student the chance to find their ideal job. In this version, Solr search engine have been introduced into this subsystem. Solr is configured to do full and delta imports from solr view (jobfairdb in MySQL database).

The uses cases related to this subsystem are:

Ø VJF-057 Advance search

Ø VJF-060 Navigation bar Search

Ø VJF-061 Reset Advance Search Inputs

Ø VJF-062 Set Job Search Status On

Ø VJF-066 Save Query

**API Subsystem**

This subsystem allows admin to manage API keys from which other services can push jobs to our system. Also, it enables the CareerPath job import, synchronization, and lets admin turn on the ability of others to push into our system.

The uses cases related to this subsystem are:

Ø VJF-058 Disable API Querying

Ø VJF-065 Import Jobs

**SMS message subsystem**

This subsystem gives the ability to send messages to students as well as allow the users to receive reminders about upcoming interviews directly in their SMS enabled phones. The SMS messaging subsystem is designed to allow this kind of functionality which can keep the users connected and active with the system.

All of this is possible by using the Twilio cloud communications service. Twilio is a company that provides services such as SMS sending and receiving, speech and text recognition, conference calling etc. The most compelling feature is that an extensive and well documented API is provided to developers, this will be necessary in order to create a system that allows for sending text messages to users, as well as validating their identities.

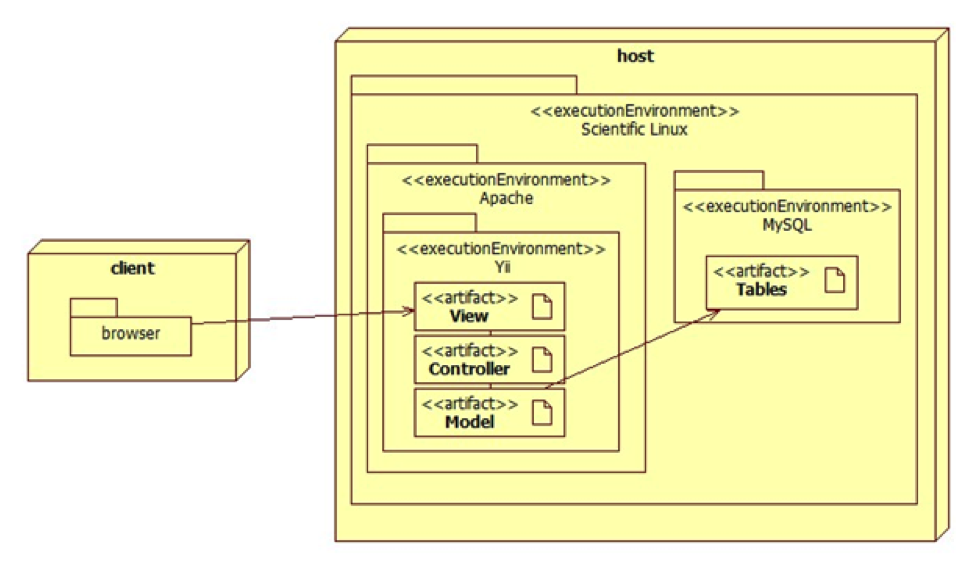
Ø VJF-045 Send SMS to student

Ø VJF-046 Send interview reminder

Ø VJF-047 Confirm phone number

## 2.3. Hardware and Software Mapping

The previous team developed a deployment diagram which indicates the mapping of hardware and software. For our Virtual Job Fair 5.0, we are still using the Model-View-Controller architectural pattern. Moreover, our team requested a virtual machine running on Scientific Linux with an Apache server and the Yii framework on the FIU SCIS network, identical to the set up used by the previous team. For that reason, the same deployment diagram as before will be used. Therefore, the description below was that done by the previous team, and remains valid for Virtual Job Fair 5.0.

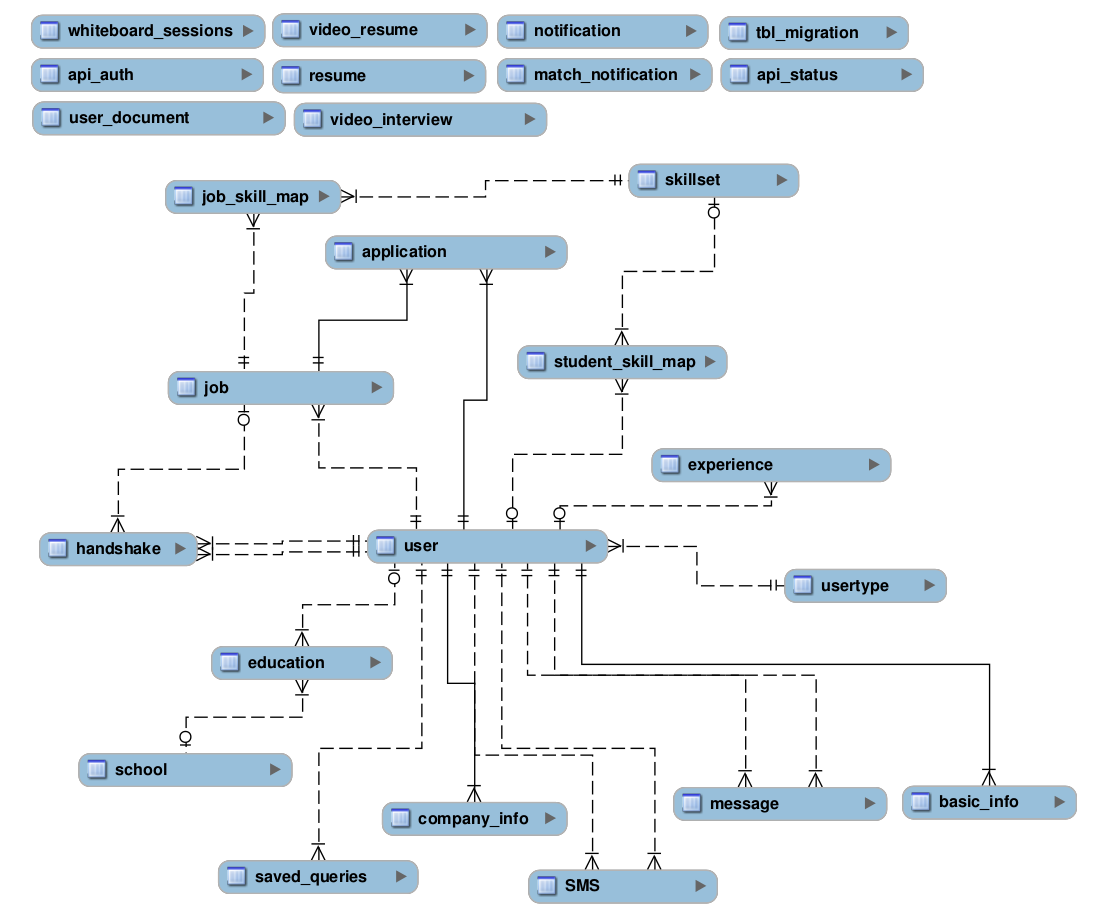


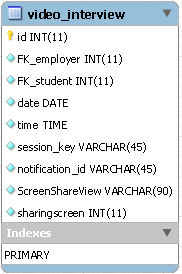
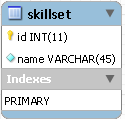
**Figure 2.3.1 Hardware and Software**

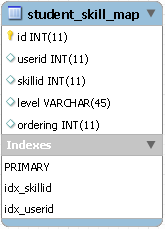
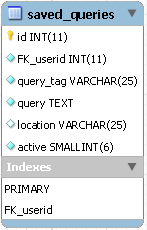
The deployment diagram shown above represents the hardware and software mapping in the Virtual Job Fair system. The main components of the system are the Apache and MySQL environment hosted on a Linux operating system. The Yii framework environment is using apache to execute, and contains our various artifacts used in development (Model, View, and Controller). The models are mapped to tables in the MySQL environment set up on the same machine. The browser on the client’s machine communicates with the server using HTTP.

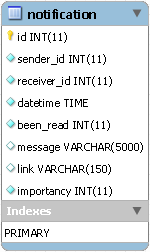
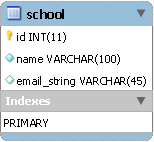
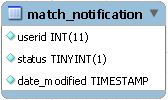
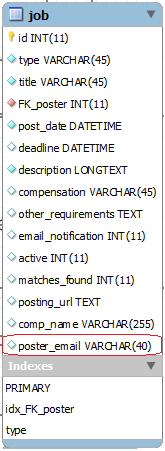
## 2.4. Persistent Data Management

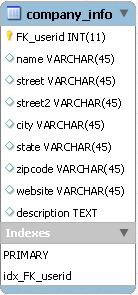
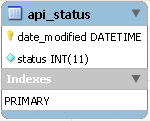
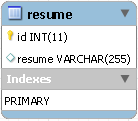
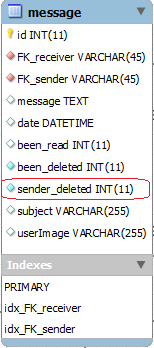
The previous group completed an ER diagram. From version 4.0 to version 5.0 there was only two minor changes to the database which I will reflect on the next screenshots. The change are the addition of the email of the employer posting a job in the job table. The name of the column in the database is poster\_email and it serve us to differentiate if a Guest Employer posted a job from another same type of user. The other change was made in the message table, were we added a column named sender\_deleted. This new column allows to avoid message dependency and correctly show the message to the two parties.

****

  ****

## 2.5. Security/Privacy

**· User password will be hashed in the database.**

Upon registration into the system, passwords entered will be hashed right away and will not be saved anywhere on the system. Upon login, the password entered again will be hashed and the hashed data will be used to query the database.

**· Administrator will be able to disable users and delete jobs**

An administrative console will be provided to a person to allow basic duties that may be needed in the future. Due to abuse of the system, it may be necessary to delete jobs or disable users.

· **Yii access control rules**

The Yii framework provides access control with respect to any controller being used. This access control will reject a subset of users (not logged, students, employers, etc…) from performing certain actions. For example, users that are not logged in will not have access to profile pages.

**· Cross-site Scripting Prevention**

The Yii framework takes measures against common web exploitations such as cross-site scripting or MySQL injection. Using Yii, we can be rest assured that such things should not occur.

**· Secure registration process**

The registration process is not as simple as most sites, especially for employers. Administrators will have to verify employers after they register to ensure they are actual employers to ensure the integrity of the system. Only then will they be able to post jobs and interact with students.

**· Privacy**

Students and Employers are distinct user types and therefore have distinct permissions. It may be necessary to allow employers to do actions that students cannot. For example, students should not be able to post a job or schedule an interview, which clearly employers should be able to. Likewise, students will only be able to view an employer’s profile and will not be able to view other student’s profiles, since it may contain information which should not be shared, such as phone number or email.

# 3. Detailed Design

The detailed design chapter introduces the system in terms of subsystems and the relationships among them. Initially, the system is decomposed into subsystem, with each subsystem described in terms of behavior and structure. Then, the static model is introduced in terms of subsystems with descriptions for each. After that, the dynamic model is presented in terms of state machine diagrams with the main control object for each subsystem. Finally, class interfaces and constraints for the main control object in each subsystem are presented.

## 3.1. Overview

**Screen Share subsystem**

Additionally the video screen Share subsystem allows users to share their screens with one another. This functionality is facilitated by the Screen Leap API. Screen Leap is a service that provides screen sharing functionality without the need for any of the users to install any specific software all that is required to function is the latest version of Java for the broadcaster. Screen Leap is able to function perfectly with the system by providing different API calls that allow for easy sharing of information. By eliminating the need for the user to download extra software we are able to facilitate and simplify the usage of the Virtual Job Fair website.

**Whiteboard subsystem**

The whiteboard subsystem will allow users to draw on a whiteboard during a live interview. The whiteboard includes drawing, writing, erasing, clearing and picture-saving functionality, which makes it an ideal tool to share images during a live interview, to jot down notes or simply to brainstorm.

**Video Interview Subsystem**

The Video Interview Subsystem allows users to have a virtual interview. This is possible thanks to Web-RTC technology. Web-RTC is an open source project that allows web browsers to communicate directly with each other with the aid of Java Scrip API calls and HTML5. This new technology makes the communication between internet users easier than traditional methods. Thanks to Web-RTC one can share video feed with other users without the need for media servers or plug-ins.

The synchronization of video interviews is handled in a table in the database. This is very important because we need to make sure only users scheduled to have an interview can be in the interview page. This is done by using a session key which is unique for each video interview scheduled. Users that arrive at the interview page would do so via a link which has many parameters such the session key. Once both users arrive at the interview page, the application will check for the session key parameter and match them, and only those users that have matching session keys would be able to connect to each other.

**E-mail Message Subsystem**

The E-mail messaging subsystem allows users to keep in contact and communicate with each other right on the system. It is very similar to a traditional inbox, only that it is internal to the system, similar to LinkedIn’s messaging. The messaging subsystem uses database tables to store and retrieve messages sent between users. It allow employers to message students and in doing so open up a line of communication with them; i.e., a student is able to message an employer only after the employer has initiated the communication with that particular student. Storing and retrieving messages efficiently is imperative to a successful messaging system. The messaging subsystem uses AJAX to rapidly access and store data; allowing users to interact faster with the system without having to wait for server calls.

**Notification Subsystem**

The subsystem is what allows users to stay up to date with the latest system interactions. The notification subsystem alerts users of any interaction by other users that might implicate them, such as a video interview been schedule for a user, or a new job post that matches a user’s skills.

The notification subsystem relies on the database structure to be able to efficiently keep the users inform. Due to the relationships between tables in the database the notification subsystem can easily detect what notification belongs to what user. Also it is important to sort each notification by category and level of importance. The notification subsystem achieves this by storing different types of notifications in the database and mapping them to their respective categories.

**Automated Notification**

This subsystem make use of the “job matching notification engine”. It is in charge of sending notification emails with the job listings to the students based on jobs matching their skills or based on customized saved query preferences. Students and employers can now get notifications in between any period of time they want. They job listing receive in the email will contain jobs from outside sources like Indeed.com, and CareerBuilder.com, as well as the job posting from the database which will contain jobs from FIU CareerBuilder. And for the employer it will be a list of the student that match the saved query description.

**Student Profile subsystem**

Being able to create a good profile fast and efficiently is very important. The profile creation subsystem takes care of this by allowing students to import profile information from third party websites such as LinkedIn. This ensures integrity of the data in students’ profiles, and makes it very easy for students to create their profiles.

This is possible by using API calls to LinkedIn and retrieving the data from LinkedIn users. As it is to expect, the user must grant permission to do this by providing his/her login credentials which are handled by the LinkedIn API.

**Merge Account**

This subsystem provides students with a view form to input the username and password of the other account. Then validate the information given, and if the information is verified, merge the two accounts by comparing the student information that he or she has in the database. And give the student user the choice to keep the information that he/she wants when a merge conflict happens.

**Linking Account**

This subsystem provides students with a way to link all third party account into one account. Also, it lets the student know if they are linked to a third party account or not, and which one. When linking, students can choose which information to keep if there is a conflict among information.

**Student job match subsystem**

Making the right connection is what this web application is all about. Therefore, an efficient algorithm to match students to job openings is very important. The student job match subsystem takes care of matching students with the required skills to job post, making the job of the recruiters easier, as it shrinks the search to only the most qualified individuals for the job.

The student job match subsystem relies on the relationships between the data in the job table. By matching job skills to students skills listed on their profile the algorithm can effectively narrow down the search to only those individuals who possess those skills.

**Search Subsystem**

The search subsystem is compose of the key features of the job search, which allows students and employers to perform advanced search, navigation bar search, as well as save queries to later receive email notifications with job results. This subsystem make the use of job search possible, giving the student the chance to find their ideal job. In this version, Solr search engine have been introduced into this subsystem. Solr is configured to do full and delta imports from solr view (jobfairdb in MySQL database).

**API Subsystem**

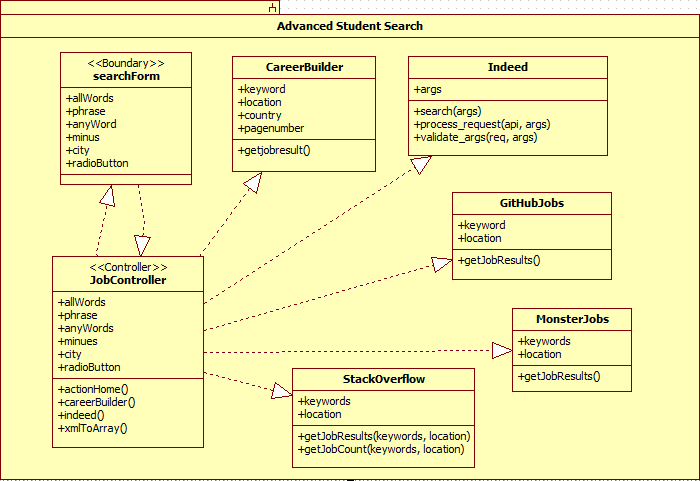
This subsystem allows admin to manage API keys from which other services can push jobs to our system. Also, it enables the CareerPath job import, synchronization, and lets admin turn on the ability of others to push into our system.

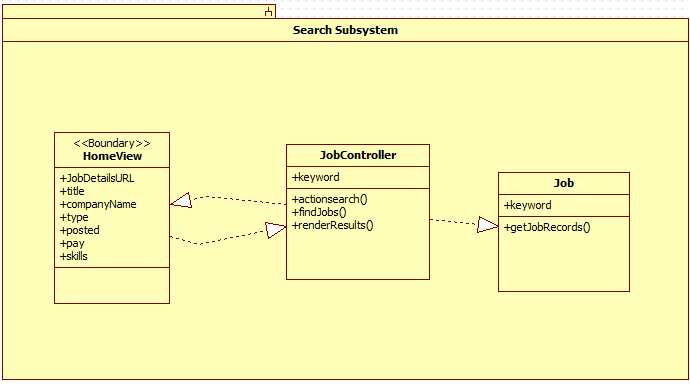
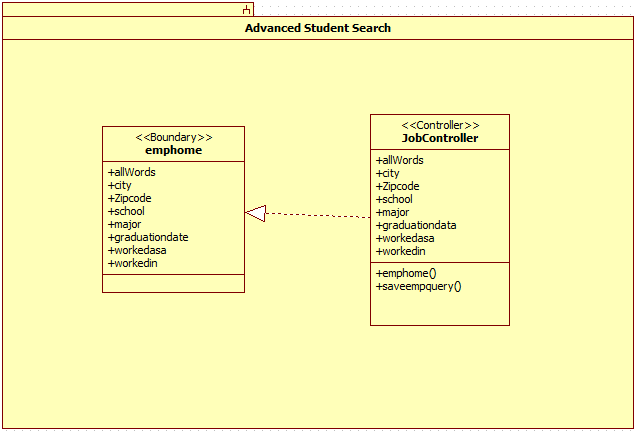
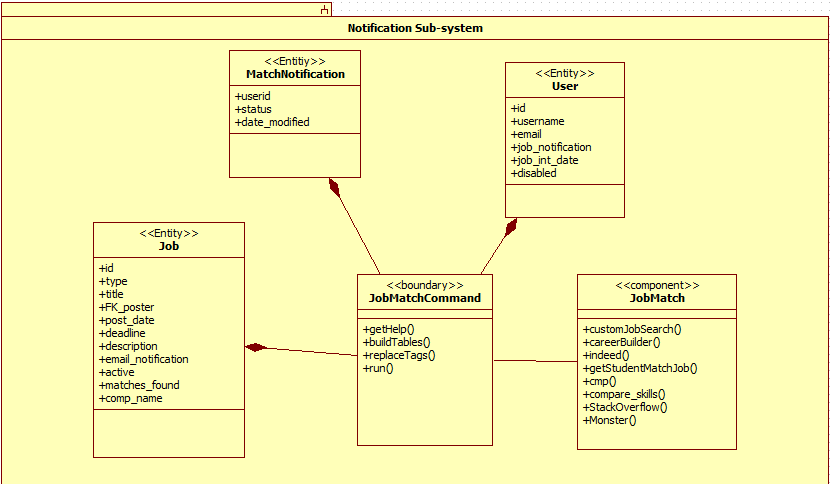
**SMS message subsystem**

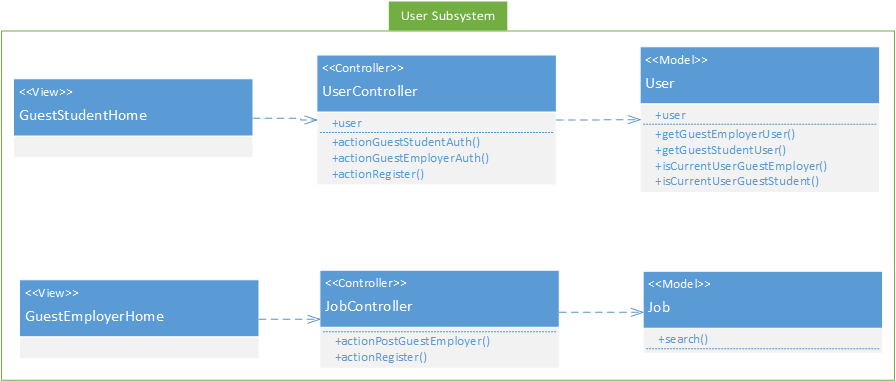
This subsystem gives the ability to send messages to students as well as allow the users to receive reminders about upcoming interviews directly in their SMS enabled phones. The SMS messaging subsystem is designed to allow this kind of functionality which can keep the users connected and active with the system.

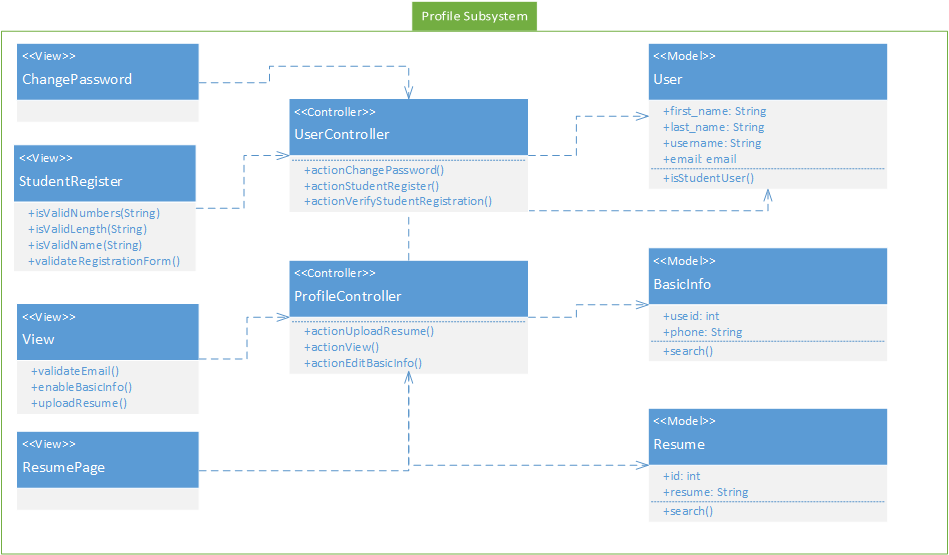
All of this is possible by using the Twilio cloud communications service. Twilio is a company that provides services such as SMS sending and receiving, speech and text recognition, conference calling etc. The most compelling feature is that an extensive and well documented API is provided to developers, this will be necessary in order to create a system that allows for sending text messages to users, as well as validating their identities.

## 3.2. Static model.





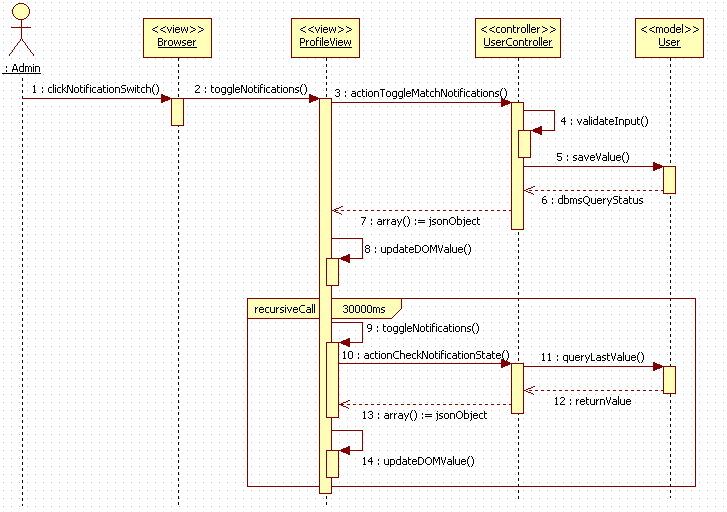




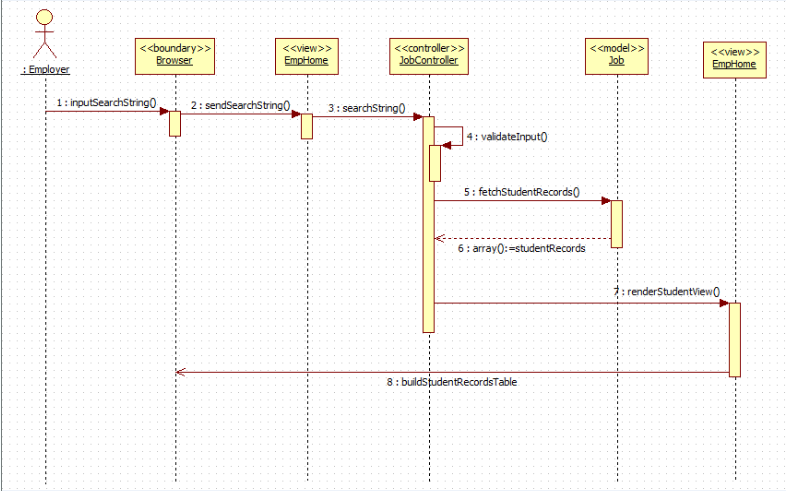
## 

## 3.3. Dynamic model – state machine diagram for the main control object in each subsystem. Include the design of the main algorithms used in the problem solution.

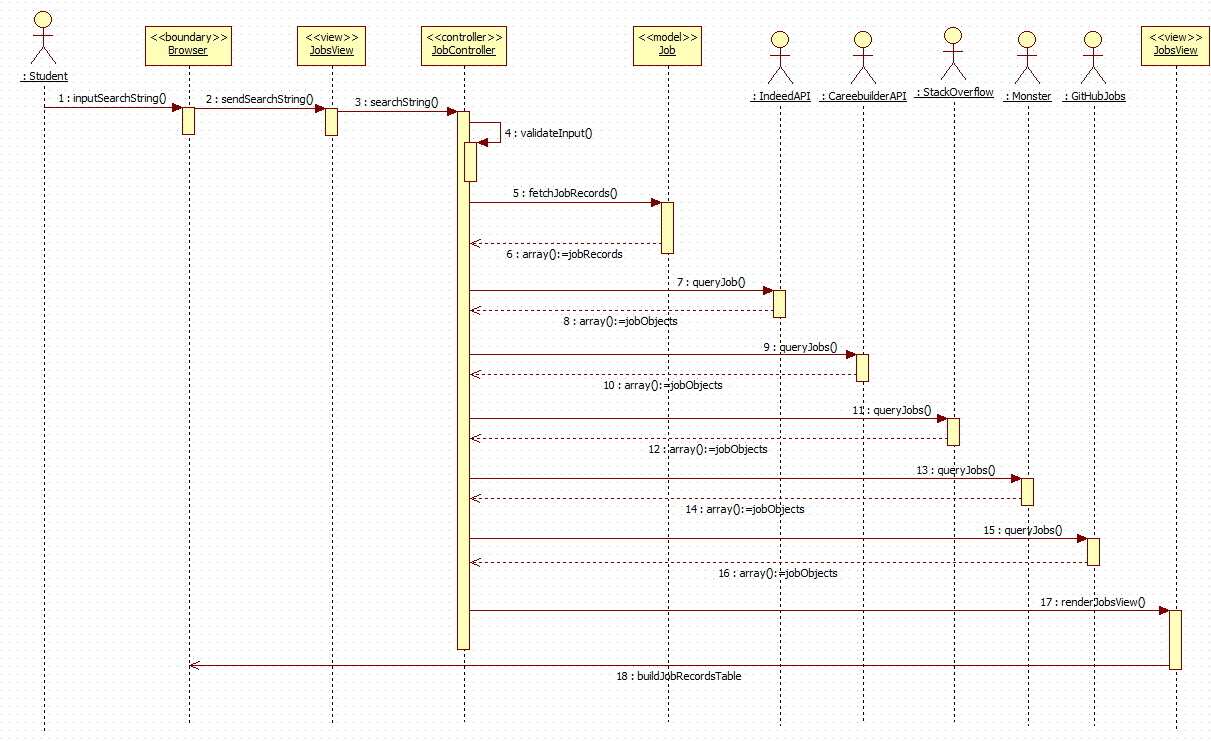
**Notification Subsystem.**

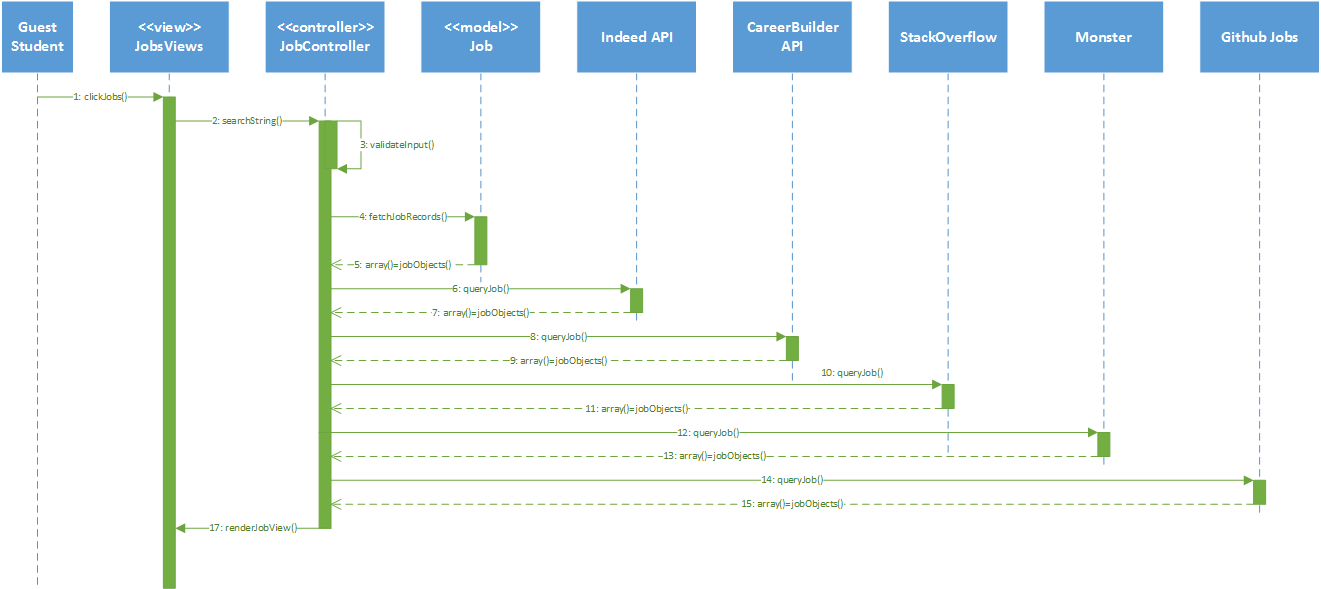


**Advanced Employer Search Subsystem**

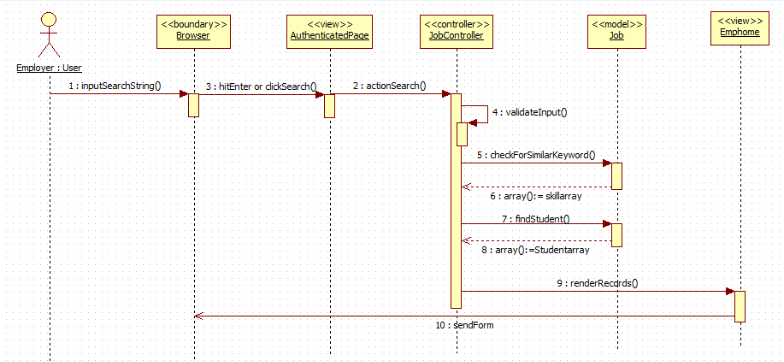


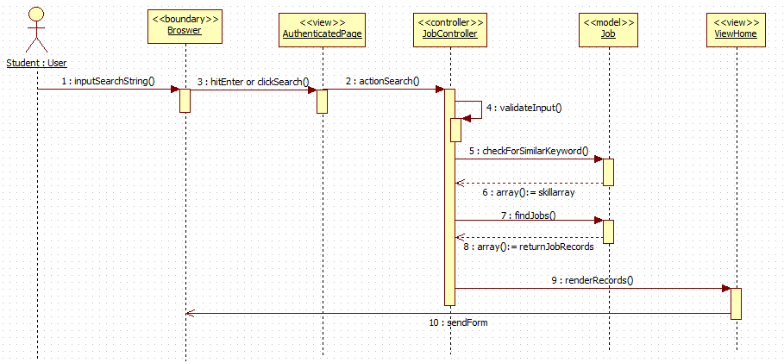
**Advanced Student Search Subsystem**



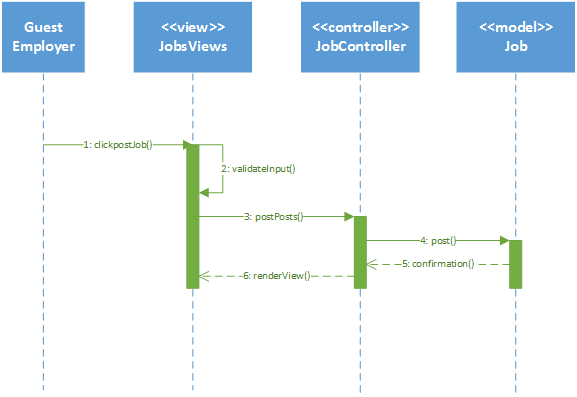


**Search Subsystem**





**Guest Employer User Job Post Sequence Diagram**



## **Student Profile Upload Resume Sequence Diagram**

## C:\Users\ralonso\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Update Resume Seq Diag.png

## **Student Profile Views Resume Sequence Diagram**

## C:\Users\ralonso\AppData\Local\Microsoft\Windows\INetCache\Content.Word\View Resume Seq Diag.png

## 3.4. Code Specification - describe the class interfaces (attributes and method signatures) and constraint (invariants, pre-condition and post-conditions) for the main control object in each system.

**Automated Notification Subsystem**

The following are some of the main methods that realize the Automated Notification service itself.

**buildTable()**

This method can be found in the JobMatchedCommand class. This method builds all the tables for the email notifications for the student, and employer. This method creates the job match search results for the student notification based on skills, or customized saved queries. Also, it creates the table for the employer notification whenever there are students matching any job posting.

**customJobSearch()**

This method can be found in the JobMatch class. This method returns an array containing job postings from Indeed.com, CareerBuilder.com, and CareerPath matching the customized saved queries each student has. The customJobSearch() method uses the Job model to obtain the jobs from the database. It also calls the indeed and careerBuilder methods.

**Search Subsystem**

The following are the main methods that realize the Advance Search and Save Query service.

**actionHome()**

This method performs the same search algorithm than **actionSearch()** on JobController.php. As of this version, **actionHome()** will delegate the search part of its algorithm to **actionSearch()**.

**actionSaveQuery()**

This method can be found in the JobController class. This method validates the search query to be saved to ensure proper the query is valid. It then uses the User model to save the customized query into the saved\_query table under the user id, with the user provided name.

**actionempHome()**

This method can be found in the JobController class. This method takes care of getting the students from the database. It validates the input search query when users perform an advanced search. It checks if the employer is selecting any queries and it searches accordingly. The return is an array that contains all the student that contains all of the criteria or a union of different searches in case the employer selects many queries.

**actionSearch()**

This method can be found in the JobController class. The role of this function is to search all the available API’s for a certain keyword and location. If is internal search it will not go to the APIs to query for jobs. If no location is specified then it will search for job postings in Miami, Florida as default.

* Indeed API
  + Currently using authorization number 9436933956285809 for the development environment and production as well. This authorization key was created under the email address [virtualjobfairfiu@gmail.com](mailto:virtualjobfairfiu@gmail.com) with password cis49112014.

# 4. Glossary

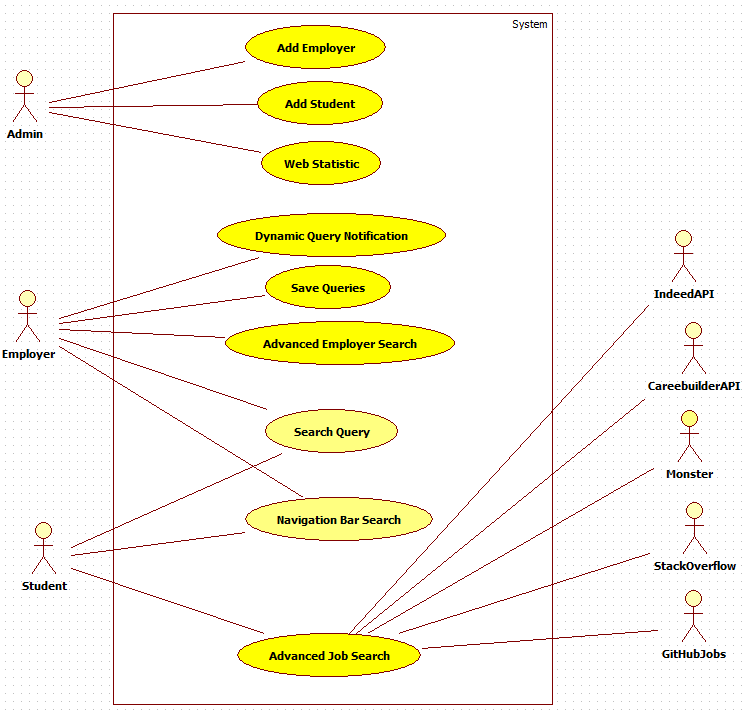
|  |  |
| --- | --- |
| Term | Meaning |
| **Class Diagram** | A pictorial representation of all the classes in the system |
| **Functional Requirement** | A function supported by the system, where a function is a set of inputs, the behavior, and outputs. |
| **Non-Functional Requirement** | A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. |
| **Object Diagram** | A pictorial representation of an instance of a class with example of how the data of the class will be populated |
| **Sequence Diagram:** | A pictorial representation of how processes operate with one another and the user during the course of a specific piece of functionality. |
| **Use Case** | List of steps defining the interaction between the user and the system to achieve a goal |

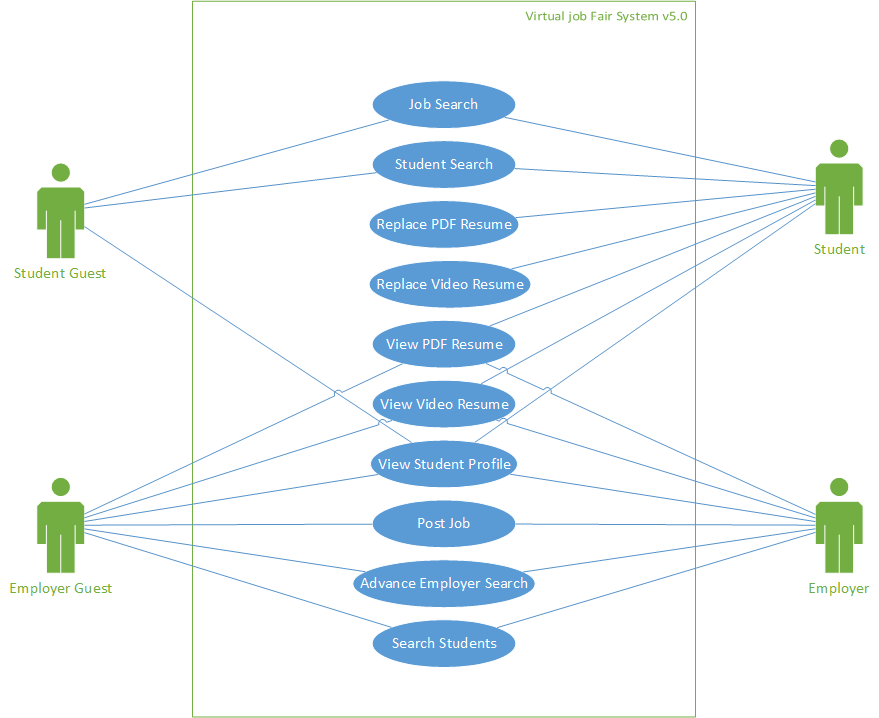
* **Salary:** a periodic payment made to an employee in exchange for services provided. Salaries are provided in yearly terms.
* **Résumé:** a document which describes a student’s qualifications, skills and education
* **Cover Letter:** a document which is used by students to introduce themselves to the companies that they are applying to. It usually goes together with a résumé

# 

# 5. Appendix

## 5.1. Appendix A - Use case diagram for use cases being implemented.





## 5.2. Appendix B - Use cases being implemented.

|  |  |
| --- | --- |
| Use Case ID | **VJF-067 Advanced Employer Search** |
| Description | Employer searches for students by skill, location, zip code, school, major, graduation date, experience, and previous positions. |
| Actor | Employer, Guest Employer |
| Pre-conditions | 1. Actor is logged in 2. Actor is on advance student search tab |
| Steps | 1. Employer clicks on the search button. 2. Employer may type one or multiple skills. 3. Employer may type one location. 4. Employer may type one zip code. 5. Employer may type one school. 6. Employer may type one major. 7. Employer may type one graduation date. 8. Employer may type experience. 9. Employer may type one position. 10. Employer clicks search |
| Post-conditions | Employer get a view to the right of the search result page with relevant students. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-068 Save Queries** |
| Description | Employer will be able to save query depending on skill, location, zip code, school, major, graduation date, experience, and previous positions. |
| Actor | Employer |
| Pre-conditions | 1. Employer is logged in 2. Employer is on advance employer advance search. |
| Steps | 1. Employer clicks on the save query button. 2. Employer may type one or multiple skills. 3. Employer may type one location. 4. Employer may type one zip code. 5. Employer may type one school. 6. Employer may type one major. 7. Employer may type one graduation date. 8. Employer may type experience. 9. Employer may type one position. 10. Employer clicks save query 11. Window of query name comes out 12. Employer must type query name 13. Employer clicks save. |
| Post-conditions | Employer gets redirected to advance search page and a confirmation or error message gets displayed. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-069 Dynamic Query Notification** |
| Description | Employer can select which query he wants to be notified about and every how much time he wants to be notified about. |
| Actor | Employer |
| Pre-conditions | 1. Employer is logged in 2. Employer is on profile page. |
| Steps | 1. Employer clicks on the student notification button (ON). 2. Employer selects one or more queries he wants to be notified about. 3. Employer selects the period of days that will make the interval that he will be notified about. 4. Employer clicks the save button. |
| Post-conditions | Employer get a view of the profile view with the updated information. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-070 Search Query** |
| Description | Employer can select which query he wants to search on the database. |
| Actor | Employer, Guest Employer |
| Pre-conditions | 1. Actor is logged in 2. Actor is on student advanced search. |
| Steps | 1. Employer clicks on the query he wants to run. 2. Employer clicks on the search button. |
| Post-conditions | Employer get a view with the results of the search. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-071 Admin Add Student** |
| Description | The administrator is able to add one user to the database. |
| Actor | Admin |
| Pre-conditions | 1. Administrator is logged in.  2. Administrator is on Home Page |
| Steps | 1. Administrator clicks on the Add Student Tab in the Administrator Dashboard 2. Administrator must add username 3. Administrator must add password and retype password fields 4. Administrator must add student first and last time 5. Administrator clicks create button. |
| Post-conditions | Administrator will get a confirmation model displayed at the right of the dashboard. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-072 Admin Add Employer** |
| Description | The administrator is able to add one user to the database. |
| Actor | Admin |
| Pre-conditions | 1. Administrator is logged in.  2. Administrator is on Home Page |
| Steps | 1. Administrator clicks on the Add Employer Tab in the Administrator Dashboard 2. Administrator must add username 3. Administrator must add password and retype password fields 4. Administrator must add employer first and last time 5. Administrator must add hide email from student option. 6. Administrator must write in about me. 7. Administrator must add company name 8. Administrator must add Location of company 9. Administrator must add city of company. 10. Administrator must add the state the company resides in. 11. Administrator must add zip code of the company. 12. Administrator must add company description. 13. Administrator clicks create button. |
| Post-conditions | Administrator will get a confirmation model displayed at the right of the dashboard. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-073 Admin Statistics** |
| Description | The administrator is able to view information about the website and database. |
| Actor | Administrator |
| Pre-conditions | 1. Administrator is logged in.  2. Administrator is on Home Page |
| Steps | 1. Administrator clicks on the website statistics tab in the Administrator Dashboard. |
| Post-conditions | Administrator will get a statistics model displayed at the right of the dashboard. |
| Exceptions | None |

|  |  |
| --- | --- |
| Use Case ID | **VJF-074 Student Search** |
| Description | The employer is able to search for a student on one skill. |
| Actor | Employer, Guest Employer |
| Pre-conditions | 1. Actor is logged in. |
| Steps | 1. Employer types one skill the search box. 2. Employer can select a skill from auto complete or fully type out the skill 3. Employer clicks submit. |
| Post-conditions | Employer is redirected to search result page with relevant students, and option to search more students. |
| Exceptions | None |

|  |  |  |
| --- | --- | --- |
| Use Case ID | **VJF-075 Job Search** | |
| Description | The student is able to search for a job on one skill. | |
| Actor | Student, Guest Student | |
| Pre-conditions | 1. Student is logged in. | |
| Steps | 1. Student types one skill the search box. 2. Student can select a skill from auto complete or fully type out the skill 3. Student clicks submit. | |
| Post-conditions | Student is redirected to search result page with relevant jobs, and option to search more jobs. | |
| Exceptions | None | |
| **Use Case ID** | | **VJF-076 View PDF Resume** | |
| Description | | Allows student actor to view his/her PDF Resume. | |
| User Story #392 | | As a Student I would like to see my PDF resume from my profile page and be able to upload a more updated one. This will benefit me as I will be able to reassure that I have uploaded a file with my resume, and it will allow me to have the most recent copy by replacing the old one. | |
| Actor | | Student | |
| Pre-conditions | | 1. Actor is logged in. 2. Actor is in his/her profile page. 3. Actor has a PDF resume uploaded in the system. | |
| Steps | | 1. Use case begins when the actor clicks on the view resume button. 2. System displays actor resume. | |
| Post-conditions | | Actor see his/her resume file | |
| Alternative Steps | | N/A | |
| Exceptions | | No resume on profile | |
| **Use Case ID** | | **VJF-077 Replace PDF Resume** | |
| Description | | Allows student actor to replace his/her PDF resume. | |
| User Story #392 | | As a Student I would like to see my PDF resume from my profile page and be able to upload a more updated one. This will benefit me as I will be able to reassure that I have uploaded a file with my resume, and it will allow me to have the most recent copy by replacing the old one. | |
| Actor | | Student | |
| Pre-conditions | | 1. Actor is logged in. 2. Actor is in his/her profile page. 3. Actor has a PDF resume to upload to the system. | |
| Steps | | 1. Use case begins when the actor clicks on the update resume button. 2. System displays a windows to select the file to be uploaded. 3. Actor selects his/her resume file and click open. 4. System stores the pdf resume and updates its path on the database. 5. Systems displays the upload was successful. | |
| Post-conditions | | Actor update his/her resume file. | |
| Alternative Steps | | 1. From step 4 the file cannot be uploaded or the path inserted in the database. 2. System displays the upload had an error. | |
| Exceptions | | 1. Invalid file format 2. Invalid file extension 3. No file selected | |

|  |  |
| --- | --- |
| **Use Case ID** | **VJF-078 Replace Video Resume** |
| **Description** | Allows student actor to replace his/her video resume. |
| **User Story #392** | As a Student I would like to see my video resume from my profile page and be able to upload a more updated one. This will benefit me as I will be able to reassure that I have uploaded a file with my resume, and it will allow me to have the most recent copy by replacing the old one. |
| **Actor** | Student |
| **Pre-conditions** | 1. Actor is logged in. 2. Actor is in his/her profile page. 3. Actor has a video resume to upload to the system. |
| **Steps** | 1. Use case begins when the actor clicks on the update resume button. 2. System displays a windows to select the file to be uploaded. 3. Actor selects his/her resume file and click open. 4. System stores the pdf resume and updates its path on the database. 5. Systems displays the upload was successful. |
| **Post-conditions** | Actor update his/her video resume file. |
| **Alternative Steps** | 1. From step 4 the file cannot be uploaded or the path inserted in the database. 2. System displays the upload had an error. |
| **Exceptions** | 1. Invalid file format 2. Invalid file extension 3. No file selected |
| **Use Case ID** | **VJF-079View Video Resume** |
| **Description** | Allows student actor to view his/her video Resume. |
| **User Story #392** | As a Student I would like to see my video resume from my profile page and be able to upload a more updated one. This will benefit me as I will be able to reassure that I have uploaded a file with my resume, and it will allow me to have the most recent copy by replacing the old one. |
| **Actor** | Student |
| **Pre-conditions** | 1. Actor is logged in. 2. Actor is in his/her profile page. 3. Actor has a PDF resume uploaded in the system. |
| **Steps** | 1. Use case begins when the actor clicks on the view resume button. 2. System displays actor video resume. |
| **Post-conditions** | Actor see his/her resume video |
| **Alternative Steps** | N/A |
| **Exceptions** | No resume on profile |

## 5.3. Appendix C – Documented class interfaces (code) for the subsystem(s) you will implement and the constraints.

**Search**

/\*

\* $allWords String

\* $phrase String

\* $anyWord String

\* $minus String

\* $city String

\* $tagName String

\*/

public function actionHome($allWords = null, $phrase = null, $anyWord = null, $minus = null, $radioOption = null, $city = null)

/\*

\* $allWords String

\* $phrase String

\* $anyWord String

\* $minus String

\* $city String

\* $tagName String

\*/

public function actionSaveQuery($allWords = null, $phrase = null, $anyWord = null, $minus = null, $city = null, $tagName = null)

/\*

\* $allWords String

\* $ZIPcode String

\* $school String

\* $major String

\* $graduationdate String

\* $workedasa String

\* $workedin String

\* $city String

\*/

public function actionEmphome($allWords = null, $city = null, $ZIPcode = null, $school = null, $major = null, $graduationdate = null, $workedasa = null, $workedin = null) {

**Automated Notification Subsystem**

/\*

\* $type String

\* $ar Array

\* $interval Integer

\*/

public function buildTable($type, $ar, $interval)

/\*

\* $query String

\* $city String

\*/

public function customJobSearch($query = null, $city = null)

## 5.4. Appendix D - Diary of meeting and tasks.

Meeting 1:

Date: 1/23/2015

Start Time: 1:00 PM

End Time: 1:55 PM

On this day I met with Instructor Masoud Sadjadi for 50 minutes from 1:00 to 1:50 and gave me guidelines about GitHub. Allowed me access the documentation and discussed the goal of the initial sprint. We tried to deploy the latest version from development to production unsuccessfully. We agreed on a new meeting on 1/26/2015 @ 9:00 AM.

Meeting 2:

Date: 1/26/2015

Start Time: 9:30 AM

End Time: 10:30 AM

On this day I met with Instructor Masoud Sadjadi and we exported Dev Database into his environment. We set up the development branch in GitHub, his local repository and tagged existing master as v4.0. We created a copy of the main.php file located under JobFair/protected/config/ and we ignore the main.php as it contains the username and password for dev and prod environments. We found a bug while deploying the project in the webserver root folder such that if the name of the folder is not set to “JobFair” the project will not function as this is hardcoded into SiteController.php under JobFair/protected/controllers/. As a result we created two new user stories as follows:

1. Making base directory configurable:
   1. As the admin of virtual job fair, I would like to be able to configure the base url. As it is now, the base url of /JobFair/ is hardcoded in many places in the code. I would like to be able to set the base directory in /protected/controllers/SiteController.php and by changing that, I should be able to install the virtual job fair under any directory that I like.
2. Making the base url configurable:
   1. As the admin of virtual job fair, I would like to be able to install this project on different places (e.g., vjf.cis.fiu.edu, vjf-dev.cis.fiu.edu, or localhost). There should not be anywhere in the code referring directly to the actual base url; instead they should refer to a variable defined in /protected/controllers/SiteController.php that specifies the base url.

Meeting 3:

Date: 1/30/2015

Start Time: 5:15 PM

End Time: 6:30 PM

Met with instructor Sadjadi and mentor Juan Caraballo and went over the user stories collected. We decided that the most important stories are the ones that improves the student and employer user directly.

Meeting 4:

Date: 2/2/2015

Start Time: 9:40 PM

End Time: 10:00 PM

Met with product owner Juan Caraballo to discuss new stories ideas (One click application process for students, administrator reporting service and on the fly resume creation). He instructed me of the policies regarding applying jobs fetched from the different API’s used by VJF. He encouraged me to follow the one click application process for jobs that are directly posted on the VJF system.

Meeting 5:

Date: 2/9/2015

Start Time: 10:00 PM

End Time: 10:20 PM

Met with product owner Juan Caraballo regarding the user stories chosen for the project. In particular he is interested in the Guest Employer and the Guest Student User. We agreed to have another meeting on Thursday, February 12, 2015 at 4:00 PM.

Meeting 6:

Date: 2/13/2015

Start Time: 5:00 PM

End Time: 5:30 PM

Met with product owner Masoud Sadjadi to present the users stories developed in Sprint 1. We agreed for Sprint 2 we should test all functionalities for the system and document the ones not performing as per the Product Owners expectations.

Meeting 7:

Date: 2/23/2015

Start Time: 6:00 PM

End Time: 6:30 PM

Met with product owner Masoud Sadjadi to check on the progress made in sprint 2.

Meeting 8:

Date: 2/27/2015

Start Time: 6:30 PM

End Time: 7:00 PM

Met with product owner Masoud Sadjadi to present the users stories developed in Sprint 2. We agreed for Sprint 3 to focus on several bug fixing to provide more stability to the system. Also, he accepted to extend the upload resume functionality to include update PDF resume, video resume and profile picture.

Meeting 9:

Date: 3/23/2015

Start Time: 9:00 AM

End Time: 10:00 AM

Met with product owner Masoud Sadjadi to present the users stories developed in Sprint 3. We agreed for Sprint 4 to focus on more bug fixing to provide more stability to the system. Also, to do a feasibility study about search engines. All with the vision to incorporate one to the VJF with a unique search bar similar to Google.

Meeting 10:

Date: 3/30/2015

Start Time: 4:00 PM

End Time: 5:00 PM

Met with product owner Masoud Sadjadi to check on the progress made in sprint 4.

Meeting 11:

Date: 4/3/2015

Start Time: 6:00 PM

End Time: 6:30 PM

Met with product owner Masoud Sadjadi to present the users stories developed in Sprint 4. We agreed for Sprint 5 to focus on fixing Indeed API and search result coming from Indeed. Also, to incorporate Solr search engine into VJF.

Meeting 12:

Date: 4/17/2015

Start Time: 7:00 PM

End Time: 7:30 PM

Met with product owner Masoud Sadjadi to present the users stories developed in Sprint 5. He advised the approach needed to make successful videos for the final presentation.

# 6. References

1. Feasibility Study & Project Plan Document v2.0. Spring 2013.
2. Feasibility Study & Project Plan Document v3.0. Summer 2014.
3. Feasibility Study & Project Plan Document v4.0. Fall 2014.
4. Requirement Document v2.0. Spring 2013.
5. Requirement Document v3.0. Summer 2014.
6. Requirement Document v4.0. Fall 2014.
7. Design Document v2.0. Spring 2013.
8. Design Document v3.0. Summer 2014.
9. Design Document v4.0. Fall 2014.
10. Final Deliverable v2.0. Spring 2013.
11. Final Deliverable v3.0. Summer 2014.
12. Final Deliverable v4.0. Fall 2014.
13. "Bootstrap." *2.2.2 Documentation*. N.p., n.d. Web. 24 July 2014. <http://bootstrapdocs.com/v2.2.2/docs/>.
14. "Documentation." *Yii PHP Framework: Best for Web 2.0 Development*. N.p., n.d. Web. 24 July 2014. <http://www.yiiframework.com/doc/>