**Database Application Programming III**

**Project Status and Design Report**

|  |  |  |
| --- | --- | --- |
| **Topic:** | Milestone 4 | |
| **Date:** | 3/4/18 | |
| **Revision:** | 2.1 | |
| **Team:** | 1. Connor Low | |
| 2. Ali Cooper | |
| **Weekly Team Status Summary:** | |  |  |  | | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | | Controller-Service Relationship | CL | 1 | | Backend Model, Business and Data services for Groups | CL | 3 | | Interfaces for Groups Business Services | Both | 1 | | Changed DB schema (added foreign keys and Group tables) | CL | 1 | | Created middleware for user authentication and authorization | CL | 1.5 | | Updated DB.ini and services to match updated database | CL | 2.5 | | Added new front end functionality | AC | 4 | | Added controllers | AC | 3 | | Added routes | AC | 3 | | Documentation | Both | 1 | |  |  |  | | |
| **GIT URL:** | <https://github.com/n4n0byte/CST256.git>  Github invite: https://github.com/n4n0byte/CST256/invitations | |
| **Hosting URL:** |  | |
| **Cast url:** | https://www.useloom.com/share/facdefaec1e64041a7eaa7c209e3967b | |
| **Peer Review:** | Y | We acknowledge that our team has reviewed this Report and we agree to the approach we are all taking. |

**Design Documentation**

**Install Instructions:**

1. Pull project from Github
   1. If using git in bash, use *git clone https://github.com/n4n0byte/CST256.git* to clone the repository into a desired folder.
   2. If using a git GUI, paste *https://github.com/n4n0byte/CST256.git* into the clone input.
   3. *Project/CLC* is the main Laravel project directory.
2. Set up database
   1. In the *Documentation* repository directory, go into */DB\_Design* and locate the *larabar.sql*.
   2. Create a schema and import *larabar.sql* tables in MySQL Workbench:
      1. Establish a connection.
      2. Create a new schema (we suggest naming it “larabar”, but you may update the *.env* file in the main project directory to reflect any name you may chose).
      3. Navigate to *Data Import/Restore* under *Management*.
      4. For *Import Options*, select *Import from Self-Contained File* and enter the location of *larabar.sql*.
      5. Set *Default Target Schema* to the name of your database schema. Start Import.
      6. Confirm that *Import of [document\_location] has finished* is logged. Test the database with a SELECT statement.
   3. Navigate to the main project directory: *Project/CLC.*
      1. Open the *.env* file
      2. Set *DB\_USERNAME=root*
      3. Set *DB\_PASSWORD=root*

**General Technical Approach:**

Naming conventions:

* **Classes**: camelcase, starting with capitals (*ClassName*).
* **URL**: lowercase with “\_” as spaces between words (*another\_uri*)
* **Everything else**: camelcase (*varName*).

Stylesheets:

All stylesheets should be compiled from .*less* or pure .*css*. The main stylesheet is *styles.less* found in the *public/css* directory.

Database connection:

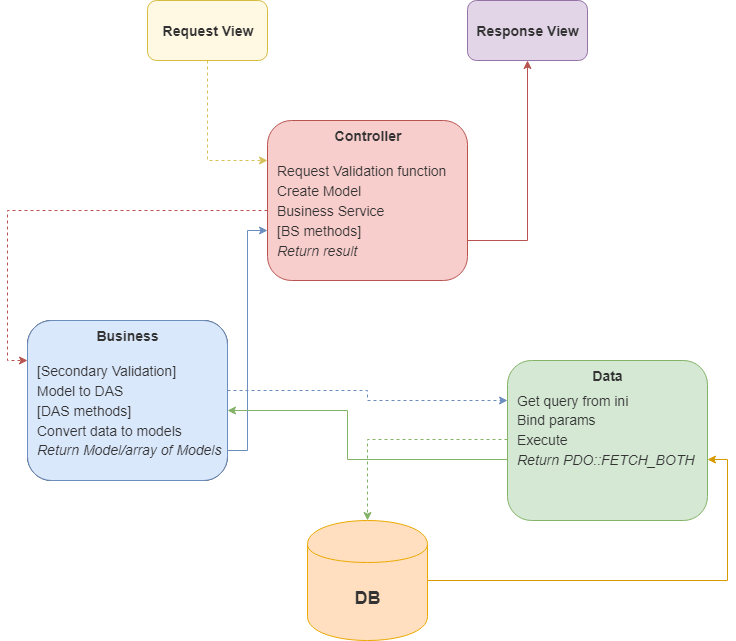
* Database, username, and password are set in .env
* Database name: *larabartest*

Git:

* Git ignore: environment and build files.

**Key Technical Design Decisions:**

**Controller-Service Relationship:**



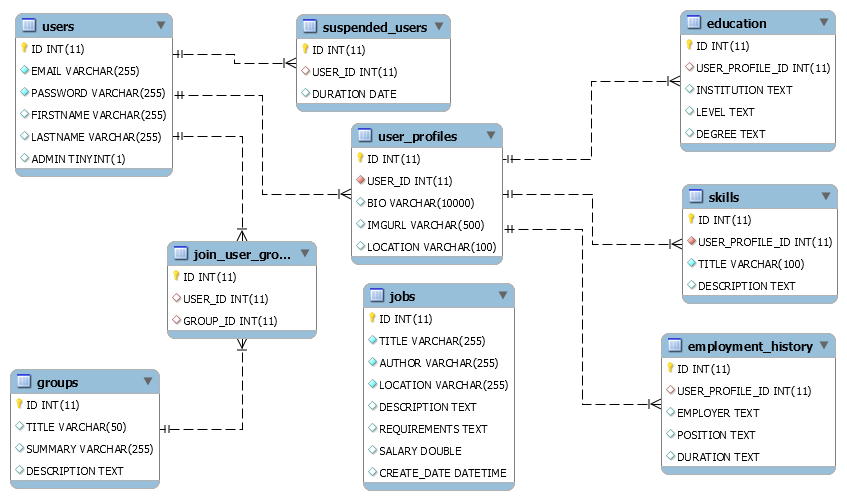
Pseudo

* Request sent: routing selects appropriate **controller**.
* Controller runs request through an appropriate Laravel validation function. If fail, return to request form.
* Request used to create a **Model**.
* Controller instantiates a **Business Service**.
* Controller calls a Business service method, passing the model as a parameter.
* If necessary, Business Service completes additional validation.
* Business Service instantiates a **Data Access Service**.
* The Business Service calls a Data Access Service CRUD method, passing the model as a parameter.
* The Data Access Service selects a query from the **Database Configuration File** (DB.ini) and passes into a PDO connection object.
* The Data Access Service stores relevant model properties into variables.
* The variables are bound to the query.
* The Query is executed and results are stored in an array using *fetchAll()* and *PDO::FETCH\_BOTH* (or boolean).
* The Data Access Service **returns** the array (or boolean).
* If not a boolean value, the Business Service converts the array into a Model or array of Models.
* The Business Service **returns** the array or Model or boolean.
* The Controller stores the results in a *$data* associative array and passes it into the next view.

Every table has matching model object, business service, controller, and data service to support the N-Layer standards. These are stored in the */src* folder.

User’s login-status is determined by the existence of a UserModel in the session.

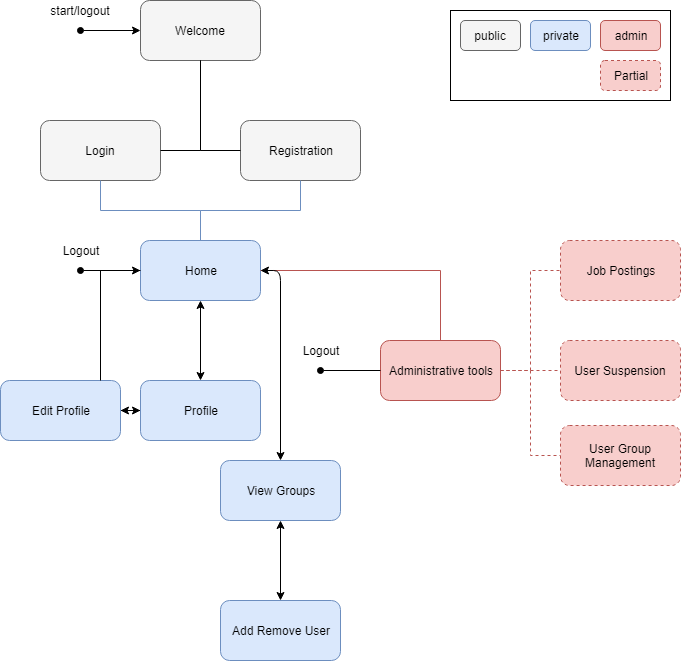
**ER Diagram:**

**

**DDL Scripts:**

https://github.com/n4n0byte/CST256/blob/master/Documentation/DB\_Design/ddl.sql

**Sitemap Diagram:**



**Security Design:**

The UserController.php currently handles form errors for Login and Registration. Actions will redirect to an error page if authentication fails.

User roles are determined by a boolean value in the database (*Admin*).

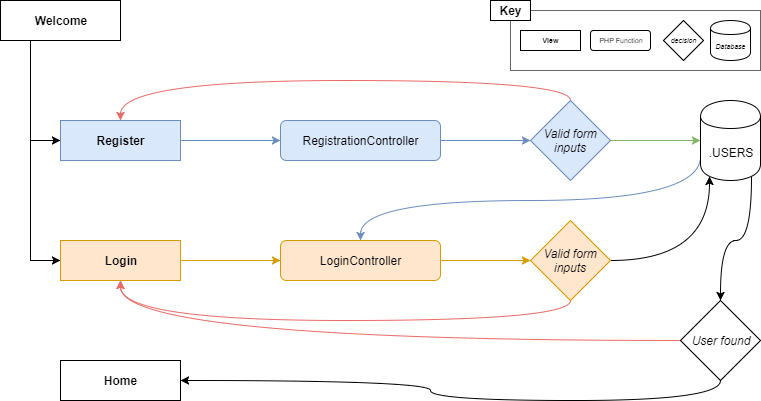
**3rd Party Interface Design:**

N/A

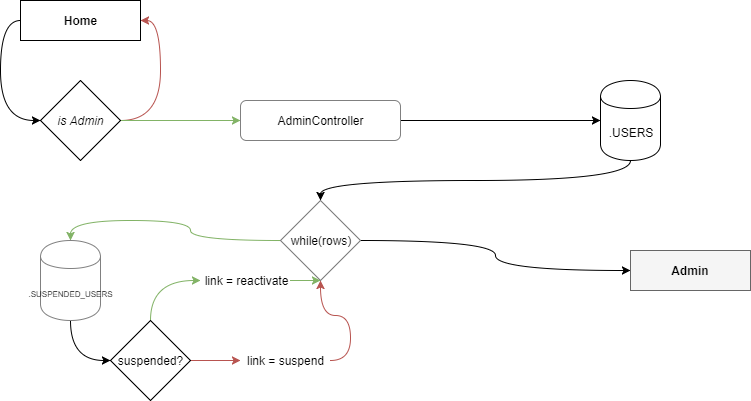
*This section should fully document any 3rd Party Service Interface API’s, how to access the service, what parameters are required by the API, and the detailed JSON data format specification that could be used by a 3rd party developer to integrate with the service and API.*

**Flow Charts:**

Authentication

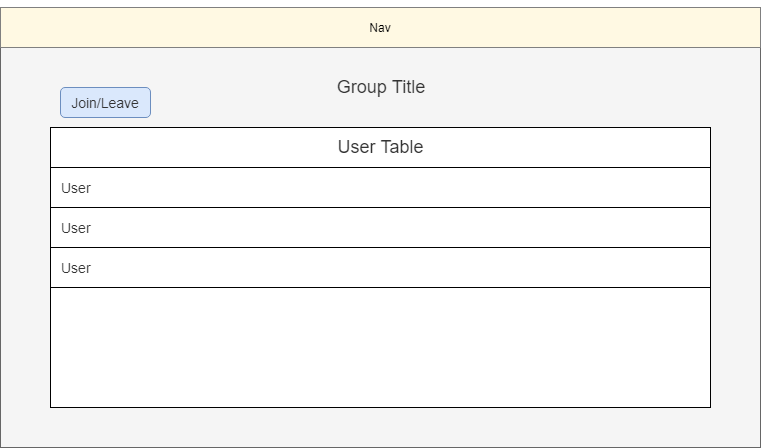


Admin: suspend/reactivate

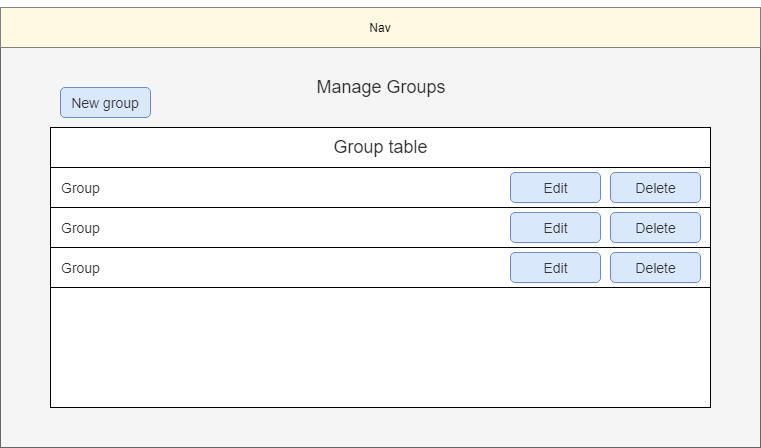


**User Interface Diagrams:**

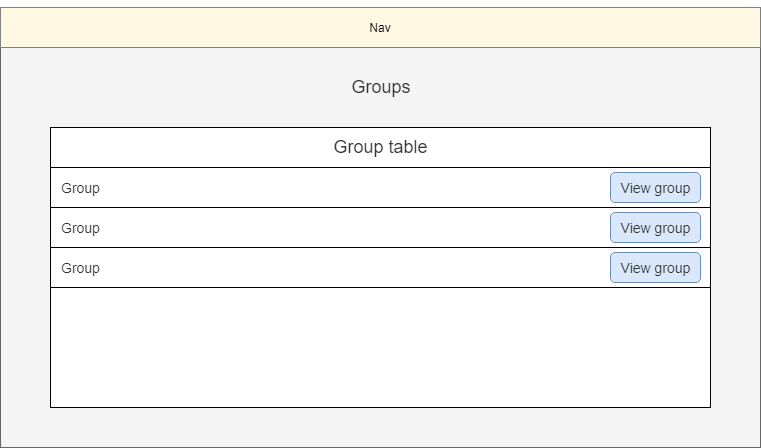
User view group



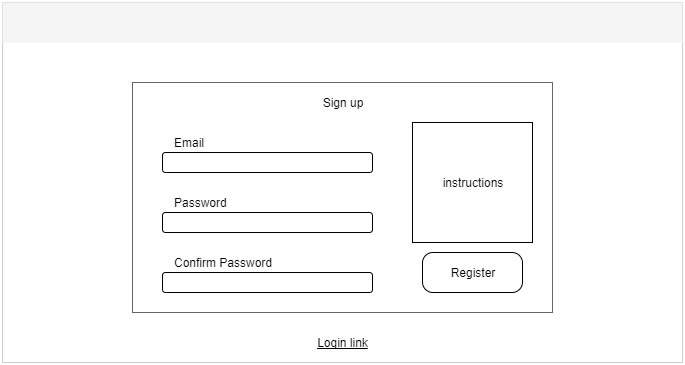
Admin manage groups



User browse groups

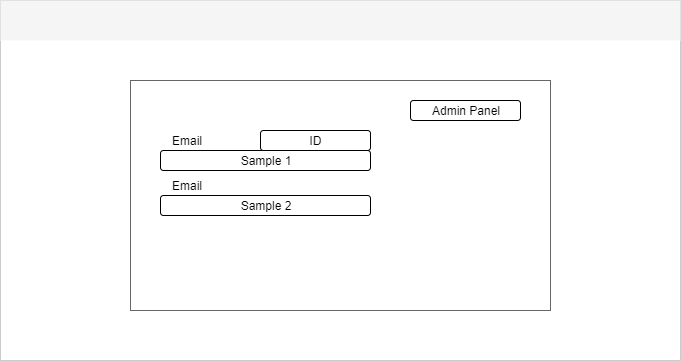


Register

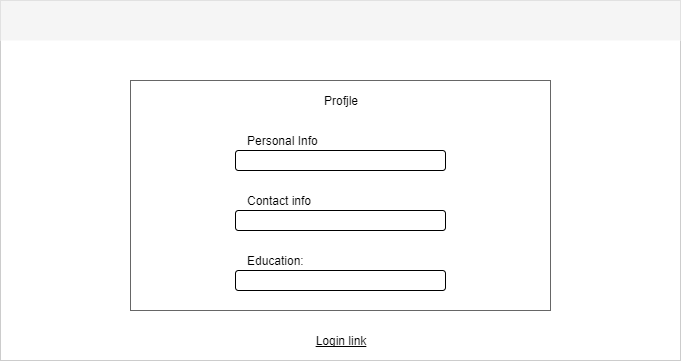


Login

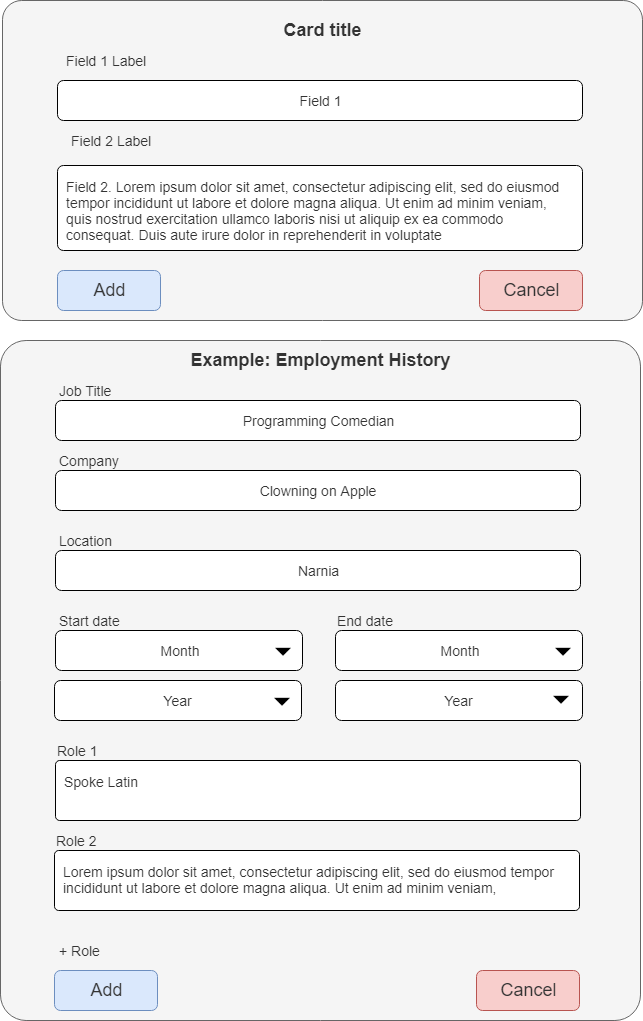




Profile

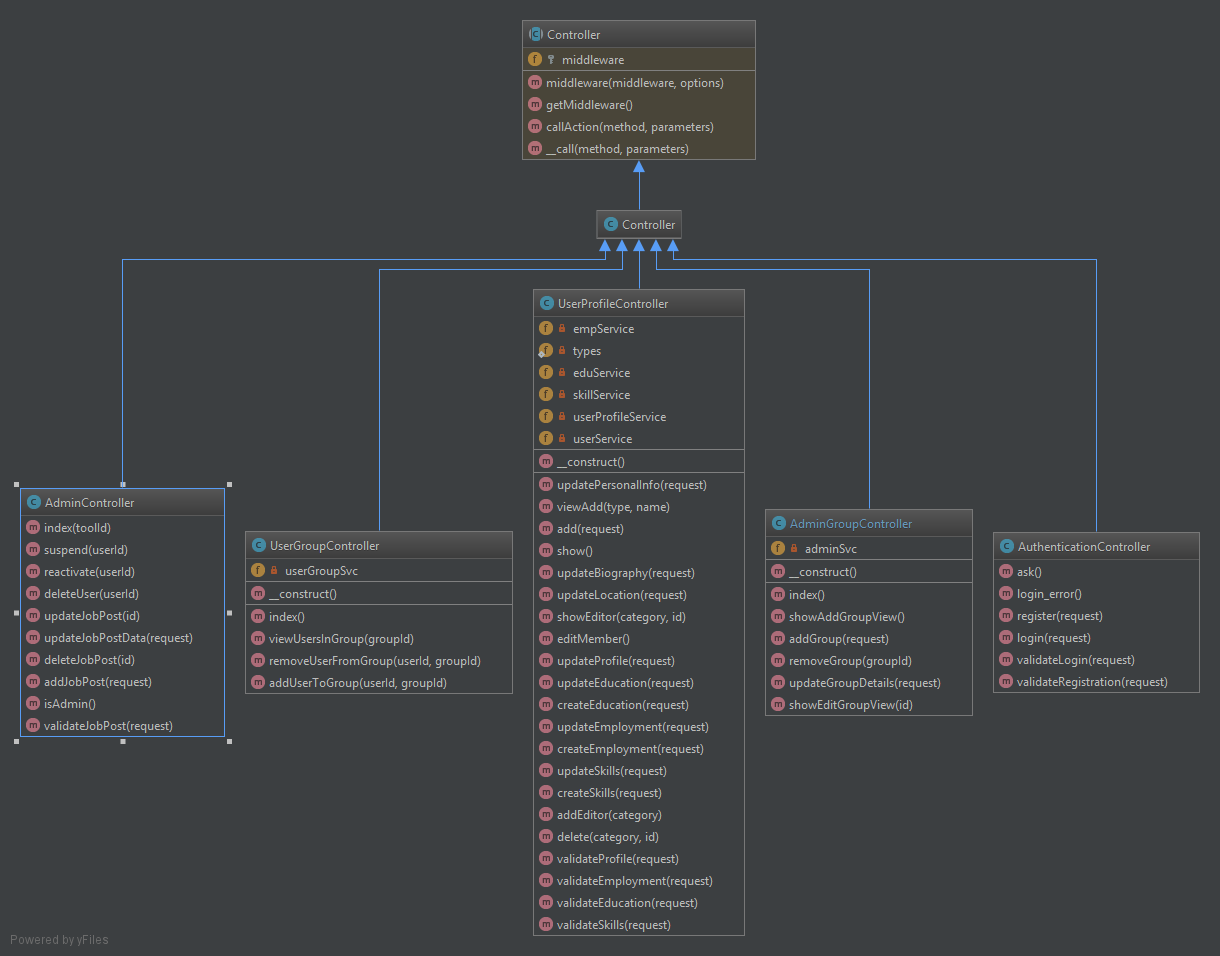


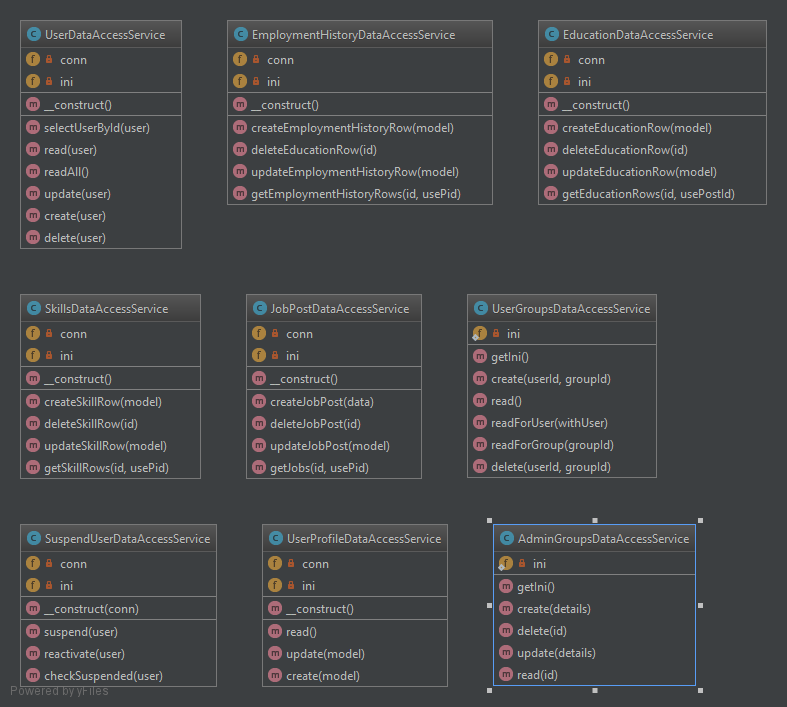
Edit Profile



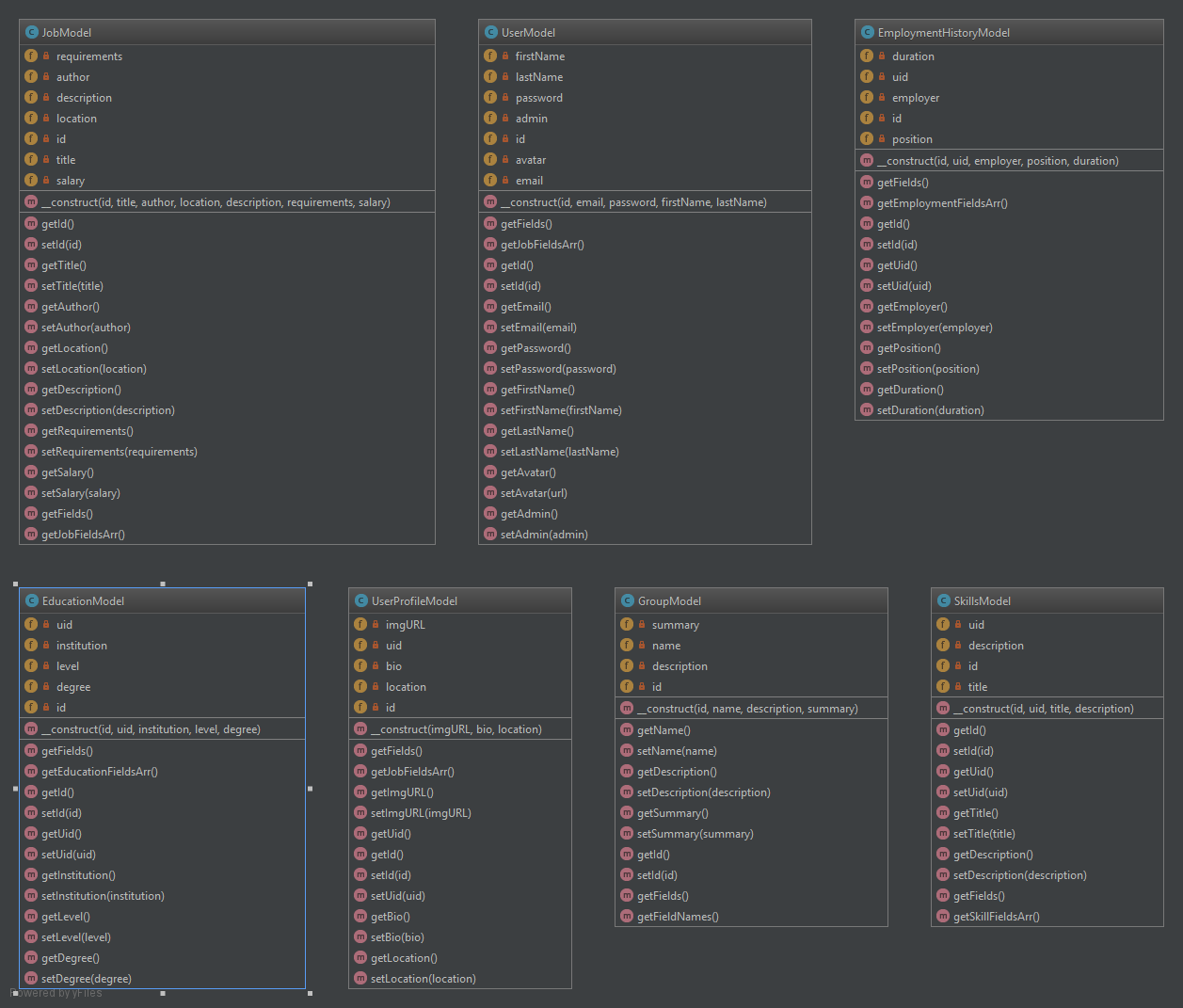
**Class Diagrams:**

Controller

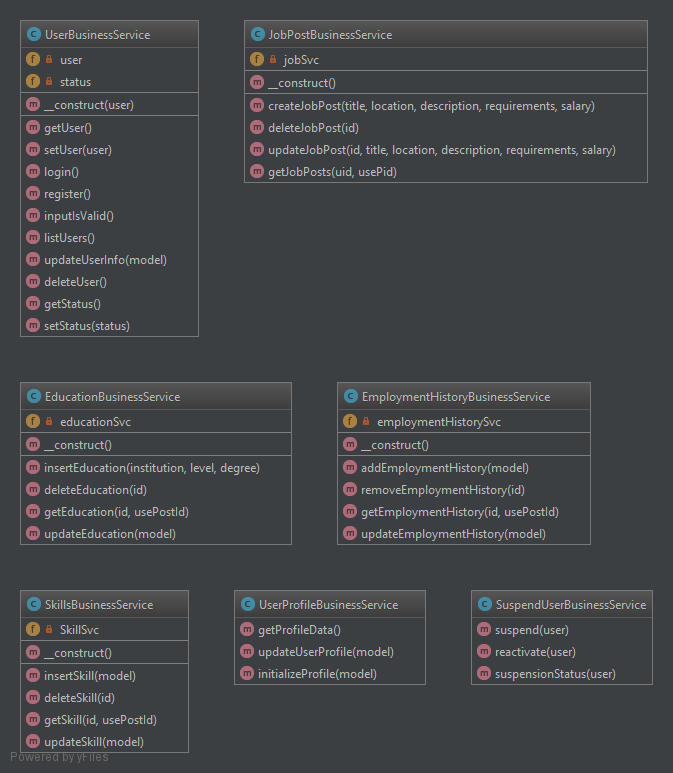


Data Access

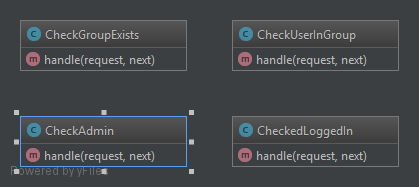
Models



Business Services



Middleware

****

**Pseudo Code:**

**MVC**

|  |
| --- |
| Controller->function  validate $Request  if false, go back to view.  $Model = new Model using $Request  $Model = new ModelBusinessService  $ResultArray = $ModelBusinessSerivce->function($Model)  $data = [$ResultArray, other objects]  return view with $data  ModelBusinessService->function($Model)  validation function($Model)  $ModelDataAccessService = new DataAccessService  $raw = $ModelDataAccessService->crudMethod($Model)  // assuming it is not a boolean  foreach($thing in $raw)  $resultArray += $Model($thing as parameters)  return $resultArray  ModelDataAccessService->function($Model)  $query = ini["Model"]["SQL Query Name"]  $vars = $Model->getVariableValues  $connection = new PDO  $connection->bindParamenters(":queryParameters", $vars)  $queryResult = $connection->executeQuery()  // assuming it is not a boolean  return $queryResult->getAssocArray() |

**Other Documentation:**

**Risks:**

* Unknown how to scaffold or auto-include models (and data services and business services) in controllers.
* No ACID with database transactions. No foreign keys are used.
* Inconsistencies in Services (naming schemes, logic nuances).
* Project structure: we have a lot of components that need to be organized,

**Known Issues :**

* No page tiles.
* Middleware protection returns 403.
* No protection against users modifying hidden form elements to change the submitted id (add middleware).
* Need to remove “Edit” nav item from user profile page.
* Most *vendor* sub-directories have *.gitignore* files that prevent the directory contents from pushing. The *vendor* directory must be copied from a pre-existing Laravel project.
* Still Bootstrap
* Particle.js pushes the scroll bar off the page for some reason, we aren’t sure.
* ~~No url protection, (page security).~~ *Fixed 3/4/18*
* Deprecated members should be removed.
* No success message for when profile is updated, or job posts.
* Editing the user profile does not auto-populate with existing information.

**Priorities:**

* ~~Assignment: Affinity Groups Administration (add a Group Name and Description, update a Group Name and Description) and Registered User (add self to Group, remove self from Group, list Users in Group)~~
* ~~Fix db stuff to work~~
* Do mockups
* Get new functionality working (Affinity Groups)
  + ~~Interfaces for new business services~~
  + ~~Create dummy and actual services~~
    - Service functionality
      * Groups by users
      * Users in Group
  + Front end
    - Admin crud for Affinity groups
  + Logger