Comments ON Service Management System (SMS)

1. Backend: [yii2](http://www.yiiframework.com/), application template: [kartik-v/yii2-app-practical-a](http://demos.krajee.com/app-practical-a), [PHPencryption](https://github.com/defuse/php-encryption)
2. Frontend: [HTML5](https://www.w3.org/TR/html5/), [CSS3](https://www.w3.org/TR/css-namespaces-3/), [AngularJS](https://angularjs.org/), [File API](https://www.w3.org/TR/file-system-api/), [SJCL](https://crypto.stanford.edu/sjcl/), [Tree-view](https://angular-ui-tree.github.io/angular-ui-tree/#/basic-example), [JSON](http://www.json.org/), [JSONC](https://github.com/tcorral/JSONC)
3. Managed Services
   1. <http://osticket.com/>
   2. <http://www.churchdb.org/>
   3. <https://www.limesurvey.org/>
4. Programming Principle: Maximize client computing for UI/UX and Data Manipulations (see data protocols).
5. UX/UI Design: Refer to lastpass.com (green tone) and onelogin.com, with responsive and modern look.
6. User Registration/Login
   1. Email verification for account activation.
   2. Password strength indicator and requirement (minimal strength must met)
   3. Capthca image verification (registration required, login optional)
   4. Password Recovery (see data protocols).
7. Manage profile
   1. Email, Password, Picture (uploaded and stored as a file with unique name).
   2. First Name, Last Name, Middle Name, Title
   3. Address: Company, Line 1, Line 2, City / Town, State / Province, Zip/Postal Code
   4. Home Phone, Mobile Phone, Fax
   5. All profile data should be stored in JSON format (including a link to the picture).
   6. Once user profile is modified, the modification will be synchronized across all enrolled services.
   7. Settings and metadata added on the fly (may not be synchronized and shown to user).
8. APP/Service list – Dynamic
   1. Services are listed in the icon-grid-view or tree-view.
   2. User can add/delete/update an external (unmanaged, form-based authentication) service by providing URL, username, and password.
   3. For integrated services, the SMS profile should be used/synchronized across all services.
   4. User can add/delete an integrated (managed, SAML 2.0/SSO) services by enrollment. Once enrolled, a service shall be removed from available integrated services.
   5. For available integrated services, clicking the item will show service description, a link to demo, and a enroll button. A service may have Frontend and Backend, for user to enroll (e.g., ticket system).
   6. Enrolment could be on an existing domain, or a new domain; could be an existing user (providing user name, password), or a new user.
   7. If an existing domain with a new user is selected, a new account in the service domain will be created, and the user profile will be copied to the service domain.
   8. If a new domain is selected, a new database (and source code) of the service will be created and started, and the user profile will be copied from the SMS to the new service domain. Church Management service is a good example for this case.
   9. A list of enrolled integrated services and external serviced will show on user’s dashboard/home page. The list can be grouped, searched, and sorted (JS client-side).
   10. Scenario Example:
       1. Support Ticket System: Existing domain, existing user
       2. Survey Management: Existing domain, new user
       3. Church Management: New domain
9. Admin
   1. Manage admin profile
   2. Add/Edit/Delete User
   3. Add/Edit/ Delete Integrated Service

Data Protocols

1. User table fields (mysql)
   1. email: CHAR(32) NOT NULL
   2. pass: VARCHAR(128) NOT NULL
   3. salt: BIGINT UNSIGNED ZEROFILL
   4. status: INTEGER UNSIGNED ZEROFILL DEFAULT 0
   5. profile: MEDIUMTEXT
   6. data: LONGTEXT
2. Security-related Functions (Confidential algorithms to be stored in a single file)
   1. di\_serv\_pass($salt, $client\_pass)
      1. Purpose: generate an encrypted password to store in SMS database based on $client\_pass and $salt.
      2. return a string of 64-128 bytes
   2. di\_match\_serv\_pass($salt, $client\_pass, $serv\_pass)
      1. Purpose: check whether the $serv\_pass matches the password produced by di\_serv\_pass($salt, $client\_pass)
      2. Return True or False.
   3. di\_gen\_key($salt, $serv\_pass)
      1. Purpose: generate a key/password to encrypt profile and data using aes256
      2. Return a string of 32
   4. di\_gen\_key\_pass($salt, $q1, $q2, $q3)
      1. Purpose: generate a key/password to encrypt/decrypt the password, giving the salt and the answers of the three security questions.
      2. Return a string of 32
3. Registration Protocol
   1. Client enters $plain\_email (JS validated), and $plain\_pass (strength enforced)
   2. Client sends $plain\_email, and $client\_pass to Server, where client\_pass = base64(sha512($plain\_pass))
   3. Server calculates $di\_email = left(base64(sha265($plain\_email)), 32), and checks $ di\_email in the User table, if the $di\_email is not in the database, creates a record with the following:
      1. email = $di\_email
      2. salt = rand() \* 18446744073709551615
      3. pass = di\_serv\_pass($salt, $client\_pass); di\_pass is a customized php function to manipulate $client\_pass using salt, and return a base64 string, e.g.:
      4. profile = aes256($key, $Json\_string), where $key = di\_gen\_key($salt, $serv\_pass), and $Json\_string contains the $plain\_email {“ct”:“…”, “lu”: “…”, “em”:”$plain\_email”}. JSON string filed name shall use two-character string, e.g., ct: create\_time, lu: last\_update, em: email, etc. A table in the database will list the long name and corresponding abbreviated name.
      5. data = aes256($key, $data), where $data is a JSON string of {“\_\_com\_angelsee\_sms\_data\_\_” : “0.0.1” }. In the future, all other client data can be appended. This item will be used for encryption authentication.
4. Login
   1. Client enters $plain\_email (JS validated), and $plain\_pass
   2. Client sends $di\_email = left(base64(sha265($plain\_email)), 32), and $client\_pass to Server.
   3. Server locates the record according to $di\_email, and verifies pass using salt according to the Registration Protocol.
   4. If successful, Server returns di\_gen\_key($salt, $serv\_pass) to Client for encryption/decryption.
5. Password Recovery
   1. Client must register a valid phone number and three security questions in order for password recovery.
      1. Birthday date (dd-mm-yyyy)?
      2. Where father was born?
      3. Where mother was born?
   2. The password recovery table contains the $di\_email, and encry\_pass = aes256($key, $json), where $key = di\_gen\_key\_pass($salt, $q1, $q2, $q3), and $json is {“\_\_com\_angelsee\_sms\_pass\_recovery\_\_” : “0.0.1”, “em”:”$plain\_email”, “ph”, “....’, “pp”: “$plain\_pass”, “st”: “...”}. (ph: phone, pp: plain\_pass, st: salt )
   3. If a client requests password recovery, s/he needs to provide email and three security questions, an email will be sent to Client with a link to a password retrieval page (email field automatically filled), a voice/sms message will be sent to Client phone with the salt. If Client enters the salt correctly on the retrieval page, the password will be displayed on the web page up to 60 seconds, and the salt will be recreated.
6. Client Profile and Data
   1. Upon successful login, Server sends an encryption key to Client.
   2. Management, Encryption and Decryption of Profile and Data are done at Client side using JS.
   3. Profile changes will be sent back to Server immediately.
   4. Data are stored locally (encrypted) first. If data changed, a save button should appear. If change happens, the whole encrypted data will be sent back to Server periodically according to profile setting (every X minutes, when browser close).
   5. To reduce data communication, a checksum of a data is stored in profile to see whether the data has been changed.
   6. If Client choose to store data on cloud storage (may be favourable to reduce Server load and increase performance), then data will not be stored and synchronized on Server (phase 2+).